**Технически Университет – Варна**

**Катедра „Компютърни науки и технологии“**

Разработка на програмна система за първична обработка на видео информация

**Дипломант:**

Иво Мирославов Янков

Факултетен №: 61160145

**Научен Ръководител:**

доц. д-р инж. Митко Митев

Съдържание

[1. Обзор на съществуващи програмни системи за обработка на видео информация 3](#_Toc460948410)

[1.1. Видео 3](#_Toc460948411)

[1.2. Видове видео обработка 9](#_Toc460948412)

[1.3. Класификация на параметри 10](#_Toc460948413)

[1.4. Операционни системи и платформи 13](#_Toc460948414)

[1.5. Класификация на програми 13](#_Toc460948415)

[2. Технологична схема за обработка 20](#_Toc460948416)

[3. Структура на програмната система 24](#_Toc460948417)

[3.1. Обща структура 24](#_Toc460948418)

[3.2 Структура на визуалния интерфейс на редактора 29](#_Toc460948419)

[3.3 Структура на сървърна част 34](#_Toc460948420)

[3.4 Манипулация на медия с FFmpeg 41](#_Toc460948421)

[4. Тестване на разработената програмна система 47](#_Toc460948422)

[5. Ръководство за потребителя 49](#_Toc460948423)

[5.1. Инсталация на нов сървър 49](#_Toc460948424)

[5.2. Използване на системата 50](#_Toc460948425)

[6. Изводи и задачи на дипломната работа 59](#_Toc460948426)

[7. Източници 61](#_Toc460948427)

[Приложение 1: Програмен код 63](#_Toc460948428)

# Обзор на съществуващи програмни системи за обработка на видео информация

Предметната област на дипломния проект са програмите за видео обработка. Проектът разглежда в детайли начини за обработка на видео информация. Анализирани са популярни и непопулярни програми от този тип. Главна цел на проекта е предоставяне на достъпна и лесна за употреба среда за обработка на видео.

Като платформа за програмата е избрана формата уебсайт, защото има най-ниски изисквания от към потребителя – не се налага инсталация на програмата и няма нужда от скъп хардуер.

# Видео

Определение

Видео е термин, който включва широк спектър от техники и технологии на записа, обработката, предаването, съхраняването и възпроизвеждането на поредица от образи в движение, със или без звук. За създаването на видео обикновено се използва видеокамера. Видеото създава илюзия за движение като използва бърза смяна на поредица от неподвижни и последователни изображения, наречени кадри. [[1](https://en.wikipedia.org/wiki/Video)] Главни характеристики на видеото са:

Брой кадри в секунда (честота на кадрите)

Човешкото око е способно да различи от 10 до 12 отделни изображения в секунда. Границата на визуалното възприятие варира в зависимост от измерваното. Когато хора гледат осветен екран, те могат да различат затъмняване, което трае 13 милисекунди. Много краткотрайна стимулация на зрителния апарат (1 ms) се възприемат от хората в продължителност на 100 ms до 400 ms поради ефекта на постоянство на зрението. Поради това краткотрайни (10 ms) осветявания в зелено, последвани от същите в червено, се възприемат като жълти от човешкото око. Ефектът на постоянство на зрението създава илюзия за непрекъснатост, което позволява поредица от неподвижни изображения да изглеждат като движещи се.

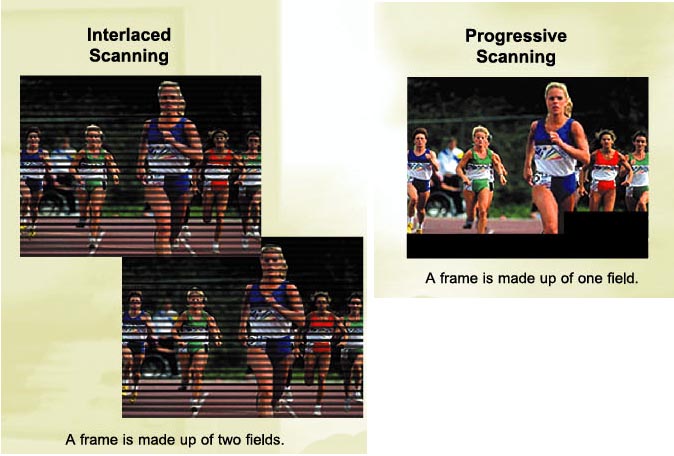
Стандарти за брой кадри в секунда са 24p, 25p, 30p, 48p, 50i, 60i, 50p/60p, 62p, 90p, 100p, 120p и други, където символите p и i обозначават формата на изобразяване (progressive или interpolated). Най-често стандартите до 30 кадъра в секунда се използват в кино индустрията. [[2](https://en.wikipedia.org/wiki/Frame_rate)]

Размерно съотношение

Описва пропорцията широчина към височина. Записва се като числова двойка, например 16:9.

Най-разпространените размерни съотношения са 1.85:1, 2.39:1, 4:3, 16:9. [[3](https://en.wikipedia.org/wiki/Aspect_ratio_(image))]

Формат на изобразяване



## Фигура 1.1: Презредов и прогресивен формат на изобразяване

Презредов формат

При този формат кадърът се разделя на няколко, като например четните редове представляват 1 кадър, а нечетните – друг. По този начин се увеличава броя на възприетите кадри от зрителя и намаля трептението по екрана.

Презредовите сигнали изискват екран, който да показва индивидуалните полета в последователен ред. Единствено електроннолъчевите тръби и плазмените екрани поддържат тази функционалност.

Един от най-важните плюсове на този формат на изобразяване е, че се спестява количество информация. При фиксиран размер на честотната лента, презредовия формат предоставя видео сигнал с двойно по-голям брой на кадрите в секунда спрямо прогресивния формат. Но това се отнася само за аналоговите и за некомпресираните видео сигнали. С навлизането на видео компресацията, която се използва във всички съвременни стандарти, презредовото изобразяване се явява неефективно.

Този формат също има и някои недостатъци. Презредово видео е създадено да бъде заснемано, съхранявано, предавано и изобразявано в един и същи презредов формат. Тъй като всеки презредов кадър е заснет в отделен момент от време е възможно бързо движещи се обекти да изглеждат размазани. [[4](https://en.wikipedia.org/wiki/Progressive_scan)]



## Фигура 1.2: Размазване на картината при презредово изобразяване

Прогресивен формат

Формат на изобразяване, съхранение или предаване на подвижни изображения, в който редовете на всеки кадър се изчертава последователно.

При този формат движението изглежда по-гладко и по-реалистично и не се наблюдават размазвания на изображението. Отделните кадри могат да бъдат използвани като самостоятелни изображения. [[5](https://en.wikipedia.org/wiki/Interlaced_video)]

Компресиране на данни

Компресиране на данните е процес от обработката на дигитални сигнали, при който информацията се прекодира, използвайки по малко битове от първоначално. Компресирането може да бъде с или без загуби.

При компресиране без загуби се откриват и елиминират статистически излишества и няма загуба на информация.

При компресиране със загуби се намалява броя на битовете като открива и премахва ненужната информация.

Степента на компресия се определя като отношението между некомпресирания и компресирания размер:

Компресирането се използва за да се намали употребата на ресурси, като дисково пространство или натоварване на мрежата при предаване на информацията. Компресираните данни трябва да бъдат декомпресирани, за да се прочетат и това поражда нужда за допълнителна обработка. Често данните трябва да могат да бъдат възпроизвеждани в хода на тяхната декомпресия, което означава че този процес трябва да се изпълнява бързо. При сложни схеми на компресация това може да породи нужда за скъп софтуер. При проектирането на схеми за компресиране се вземат предвид множество фактори, сред които: степен на компресиране, количество на изкривяване на информацията при наличието на загуби, изчислителни ресурси, нужни за декомпресиране на данните и др. [[6](https://en.wikipedia.org/wiki/Data_compression)]

Компресиране без загуба

Алгоритмите за компресиране без загуба обикновено използват методи за премахване на статистическото повторение за да представят данните в по - сбит вариант, без да се губи информация. Компресирането без загуба е възможно тъй като в повечето данни от реалния живот има статистическо излишество. Например в едно изображение може да има области в които цветът не се променя върху няколко пиксела, вместо да кодираме „червен пиксел, червен пиксел, ...”, информацията може да се представи като „279 червени пиксела”. Това е базов пример за това как работи run-length encoding, има много схеми по които може да се намали размер на файл, като се елиминира излишеството в информацията.

[LZ](https://bg.wikipedia.org/wiki/LZ) (Lempel-Ziv) комрпесията е сред най- популярните алгоритми за компресия без загуба. DEFLATE е вариация на [LZ](https://bg.wikipedia.org/wiki/LZ) с оптимизирана скорост на декомпресия и добър коефициент на компресия, но самото компресиране може да бъде бавно. DEFLATE се използва в PKZIP, Gzip и [PNG](https://bg.wikipedia.org/wiki/PNG) форматите за компресия. LZW (Lempel-Ziv-Welch) се използва в [GIF](https://bg.wikipedia.org/wiki/GIF) изображенията. LZR (Lempel-Ziv-Renau) алгоритъма служи за база при [ZIP](https://bg.wikipedia.org/wiki/ZIP) компресията. LZ методите използват таблично-базиран модел за компресия, записите в таблицата представляват повторяеми низове от данни. При повечето [LZ](https://bg.wikipedia.org/wiki/LZ) методи, таблицата се генерира динамично от предходно постъпилите входни данни. Самата таблица много често използва Huffman енкодинг (например SHRI, LZX). LZХ е кодираща схема (базирана на [LZ](https://bg.wikipedia.org/wiki/LZ)), която се използва в CAB формата на Microsoft.

Най-добрите модерни методи за компресия без загуба използват вероятностни модели, такива като прогнозирането по частично съвпадение.

Класовете от граматично-базирани кодове набират популярност, защото те могат да компресират изключително добре многократно повтарящи се текстове, например колекция от биологични данни за едни и същи или родствени видове, документ с огромен набор от версии, интернет архиви и т.н. Основна задача на граматичното кодиране е изграждането на контекстно независима граматика произтичаща от единствен низ. Sequitur и Re-Pair са практични граматични алгоритми за компресия за които има достъпни публични кодове.

Компресиране с загуба

Компресирането на данни със загуба е обратното на компресирането на данни без загуба. В тези схеми някои загуби на информация е приемливо. Изпускането на несъществен детайл от източника на данни може да запази място в хранилището. Схемите на компресирането на данни със загуба са изпълнени с изследване за това как хората възприемат данните, които са под въпрос. Например човешкото око е по-чувствително към проницателни видоизменения на яркостта, отколкото към видоизменения на цвета. Компресирането на [JPEG](https://bg.wikipedia.org/wiki/JPEG) изображение работи в частност от закръглянето на несъществени битове от информация. Има съответстваща замяна между загубата на информация и размера на намалението.

Компресирането на изображения със загуба на данни може да бъде използвано от цифровите камери за увеличаване на капацитета на съхранение с минимално понижение на качеството на снимката. По същия начин [DVD](https://bg.wikipedia.org/wiki/DVD) устройствата използват MPEG-2 видео кодеци със загуба на данни за видео компресиране.

При аудио компресирането със загуба на данни методите за психоакустика са използвани за премахването на такива компоненти на сигнал, които не могат да се чуват (или са по-слабо чуваеми). Компресирането на човешката реч е често изпълнявано с дори по-специализирани техники. Кодирането на речта или на гласа е понякога разграничавано като отделна дисциплина от аудио компресирането. Различни аудио и речеви стандарти за компресиране са описани като аудио кодеци. Компресирането на гласа е използвано от интернет телефонията, например аудио компресирането е използвано за пренасяне на аудио информацията от [CD](https://bg.wikipedia.org/wiki/CD) на твърд диск и е декодирано от аудио плейъри.

Компресиране на видео

Видео комресирането използва съвременни техники на кодиране, за да се намали изличието на видео информация. Повечето алгоритми и кодеци комбинират пространствена компресия на изображенията и времева компресия на движението. В практиката, повечето видео кодеци използват и техники за аудио компресия.

Повечето алгоритми за видео компресиране са със загуба. Некомпресирано видео изисква висока скорост на прочитане на данните (data rate). Въпреки че кодеците за видео компресиране без загуба средно работят със степен на компресиране 3, типичен MPEG-4 със загуба може да постигне степен между 20 и 200. Както при всички компресирания с загуба, се плаща цена в вида на качество на видеото, обработка при компресиране и декомпресиране, и изисквания към компютърната система. Във видео, което е компресирано прекалено много може да се наблюдават смущения в картината.

Видео информацията може да се представи като поредица от неподвижни кадри. Последователността на кадрите притежава пространствено и времево изличие, което алгоритмите за видео компресиране се опитват да премахнат, или да кодират в по-малко пространство. Повторения в кадрите могат да се съхранят като разлики в отделните кадри или като се използват ограниченията на човешкия зрителен апарат. Един от най-добрите методи за компресиране на видео е interframe компресиране, при което се използват един или повече от предишните кадри, за да се кодират сегашните. Това се постига като се сравнява всеки кадър с предходния. Ако кадърът съдържа области, в които нищо не се е променило, системата копира дадената част от предходния кадър. Ако области от кадъра са променени малко в сравнение с предишния, се декомпресорът премества, завърта, осветява или затъмнява копираната област от предходния кадър. Тази обработка се кодира с команди. Този вид компресиране е подходящ, когато видеото е предназначено единствено за гледане от потребители, но поражда проблеми, ако се наложи видеото да се обработва.

Понеже interframe компресиране копира информация от един кадър в друг, ако оригиналният кадър се повреди (настъпи грешка при прехвърляне на данните), следващите кадри не могат да бъдат възпроизведени както трябва. Някои видео формати, например DV, компресират всеки кадър независимо, чрез intraframe компресиране. „Изрязване“ на intraframe-компресирано видео е почти толкова лесно, колкото при некомпресирано видео: открива се началото и края на всеки кадър и се копира бит по бит. Друга разлика между interframe и intraframe компресирането е че при intraframe системите, всеки кадър използва сходно количество информация. При interframe в някои кадри не може да се копира информация от предходни, и изискват многократно повече информация за да се съхранят. [[7]](http://en.wikipedia.org/wiki/Codec)

# Видове видео обработка

Процесът на манипулиране на видео информация се нарича видео обработка. Този процес включва изрязване на части, преподреждане на клипове, добавяне на преходи и други пространствени ефекти.

Линейна видео обработка, с използване на видео лента

Линейната видео обработка е процес на селектиране, подреждане и променяне на изображения и звук в предварително установена последователност. Без значение от вида на камерата, с която е снимано, изображенията се обработват строго последователно. В днешни дни софтуерът за видео обработка е заменил линейната обработка.

Преди навлизането на компютрите в света на видеото, „линейна видео обработка“ се е наричала просто „видео обработка“

Нелинейна видео обработка, с използване на софтуер

Нелинейна система за обработка е система, която извършва видео или аудио не-деструктивна обработка върху входния материал. Извършва се когато информацията е съхранена във файлове а не върху макари или касети.

Нелинейната обработка позволява директен достъп до всеки видео кадър от даден клип, без да се налага търкане и преместване на съседни кадри за да бъде достигнат желания кадър. При нелинейна обработка входния материал не се унищожава, както е при линейната обработка. [[8](https://en.wikipedia.org/wiki/Video_editing)]

# Класификация на параметри

Входни данни

Съвременните програми за видео обработка приемат разновидни входни данни, които сред които: импортиране на файл от паметта или от външен носител, отваряне на файл от интернет, заснемане на видео с камера и други. Съществуват множество видео формати, като най-популярни са: avi, mpeg-2, mpeg-4, wmv, webm, flv и други.

Някои програми се нуждаят от предварително инсталирани кодеци, за да могат да отворят дадени файлови формати.

Кодек е компютърна програма, която кодира или декодира цифров даннов поток или сигнал. Използват се за кодиране на сигнали при предаване, съхранение или за декодиране при възпроизвеждане и обработване. Кодеците се използват в програми за видеоконференции, предаване на медия и видео/аудио обработка.

Изходни данни

Много програми за видео обработка създават проектов файл. Всяка програма работи със собствен формат на такъв тип файл. Например Sony Vegas работи със формат veg, докато Windows Movie Maker със mswmm.

Процесът на генериране на видео файл се нарича рендиране. При този процес се създава един цялостен клип, който може да бъде разпространяван, записан на външен носител, или съхраняван. Рендирането на практика е синонимно на процеса на компилиране в програмирането.

Подобно на входните данни, изходните данни също покриват широк набор от формати. Програмите за видео обработка предлагат множество настройки за файловете, които се генерират, сред които: избор на формат, резолюция, bit rate, кадри в секунда, aspect ratio, color space, промяна на метаданни и други.

Bit rate, е броя на битове възпроизведени за дадено време. Това е важна характеристика при видео (както и при аудио) обработката, защото определя качеството и размера на файла. Например, 3 минутно видео, записано с 1000 kbps (килобитове в секунда) ще бъде с много по-ниско качество от такова, записано с 6500 kbps, защото ще притежава по-малко количество информация. По-високото качество ще прави и файла по-голям.

Възможно е файлове да бъдат с постоянен bit rate (CBR) или с променлив bit rate (VBR). [[9](https://www.custcenter.com/app/answers/detail/a_id/4506/~/what-does-render-mean?)]

Метаданните с структурирана информация за файла. Те са се намират в началото на файла и включват информация за автора, авторските права, ключови думи, медиен формат, участващи личности, местонахождение, настройки на субтитри и много други.

Автоматично генерирани са метаданни генерирани от софтуер или от записващото устройство. Съхраняват информация за модела на камерата, с която е заснето видеото, час и дата, координати от вградено GPS устройство и други. Стандартния формат е Exchangeable Information File Format (EXIF). Повечето EXIF данни не могат да бъдат променяни, с изключение на данните за час/дата и данните от GPS-а.

Ръчно въведени са метаданни, които съдържат информация за съдържанието на видеото. Това могат да бъдат различни видове текстови описания, ключови думи и др. Тези данни могат да бъдат прочитани от компютрите и се използват от търсачките.

Повечето от метаданните са автоматично създадени, но все метаданните, писани на ръка стават все по-важни. В метаданните има голям потенциал от маркетингова гледна точка. Например използването на сайтове като YouTube не би било същото ако не съществуваха описанията и ключовите думи от метаданните, които позволяват на потребителите да откриват търсените от тях клипове. [[10](http://www.videouniversity.com/articles/metadata-for-video/)]

Инструментални средства

Инструменталните средства предлагани от програмите за видео обработка биват за промяна на отделните кадри и за промяна на потока (последователността) на кадрите.

Инструментите за промяна на отделните кадри позволяват промяна на светлината и цветовете, добавяне на специални ефекти и други. Към тази група спада известния ефект на „зеления екран“, при който даден цвят се заменя с друга картина или видео. Този похват е широко използван в киното.



## Фигура 1.3: Използване на техниката „зелен екран“

Техниките за обработка на отделни кадри са сходни с тези за обработка на статични изображения, като основната разлика, че се прилагат върху множество кадри едновременно.

С инструментите за обработка на потока кадри е възможно разделяне на видео клип на части, слепване на отделни видео клипове, добавяне на текст или изображения върху видео картината и други.

Най-често съществува времева лента, върху която се нареждат видео клиповете. Възможно е да съществуват множество слоеве, които позволяват наслагване на няколко видеа едно върху друго.

# Операционни системи и платформи

Съществуват програми за видео обработка, които са предназначени за различни платформи: десктоп, мобилни устройства, уеб услуги и др, както и за различни операционни системи – windows, ios, android, unix-базирани. Десктоп машините в повечето случаи имат по-добри характеристики от мобилните, което позволява за направата на много по-сложни програми за обработка на мултимедия, които разполагат с повече възможности. Поради тази причина най-популярните такива програми са направени именно за Windows, Linux и MacOS. От друга страна програмите за мобилните устройства са много по-опростени и имат елементарен интерфейс, което позволява бързо и лесно редактиране, използвайки базови функционалности и предефинирани филтри.

# Класификация на програми

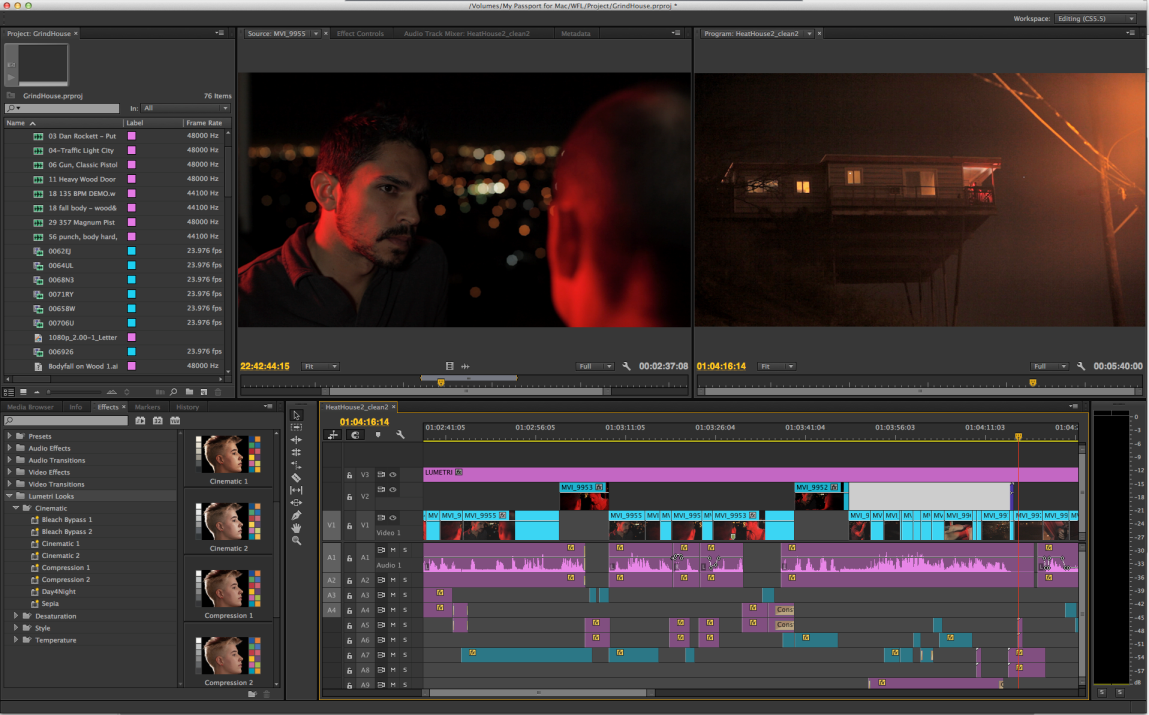
Adobe Premiere Pro

Adobe Premiere Pro е софтуер за видео обработка, базиран на времеви ленти. Част е от [Adobe Creative Cloud](http://en.wikipedia.org/wiki/Adobe_Creative_Cloud), което включва програми за видео обработка, графичен дизайн и уеб разработка.

Premiere Pro се използва от телевизионни оператори като BBC и CNN.

Premiere Pro поддържа обработка на видео с висока резолюция до 10,240 × 8,192, в до 30-бита за цветови канал. Архитектурата на програмата позволява да се импортират и експортират множество формати от файлове и кодеци. Работи на MacOS и Windows. Когато се комбинира плъгини от Neo серията на Cineform, поддържа 3D обработка с възможност за преглеждане на 3D материали, използвайки 2D монитор, като прави индивидуални корекции за лявото и дясното око.

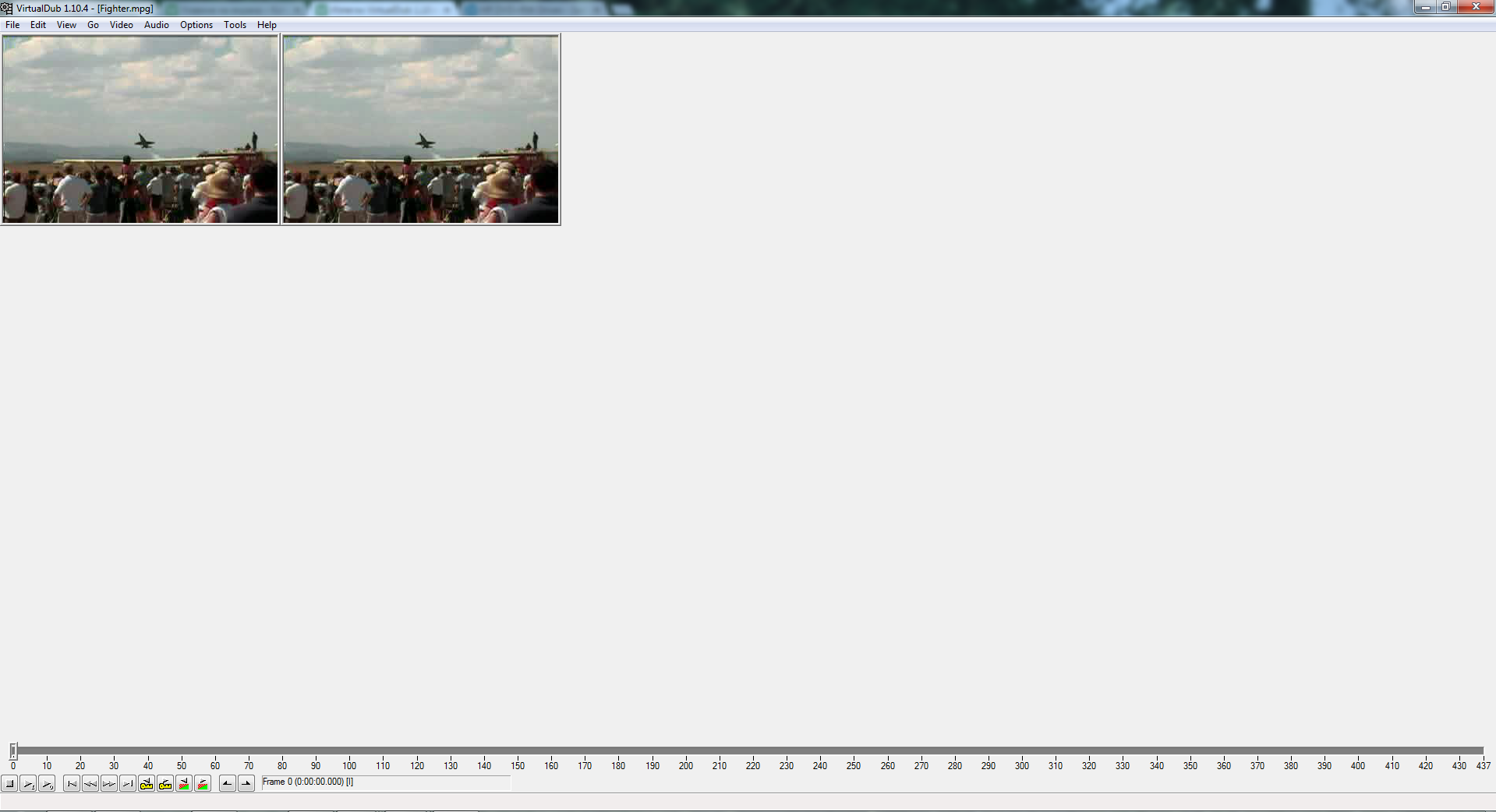
Поддържани видео файлови формати са ASF, AVI, DV, DNxHD, F4V, GIF, M1V, M2T, M2TS, M4V, MOV, MP4, MPEG, MPE, MPG, M2V, MTS, MXF, VOB, WMV и др. [[11](https://helpx.adobe.com/premiere-pro)]



## Фигура 1.4: Adobe Premiere Pro

VirtualDub

VirtualDub е програма за видео заснемане/обработване за 32-bit и 64-bit Windows платформи (98/ME/NT4/2000/XP/Vista/7). Не притежава всички възможности за обработка, каквито има, например, програмата с общо предназначение Adobe Premiere, но за сметка на това е специализирана за бързи линейни операции върху видео. Притежава възможности за batch обработка на голям брой файлове и може да бъде разширявана с видео филтри от трета страна. Създадена е главно за обработка на AVI файлове, въпреки че може да чете (без да записва) MPEG-1 и може да работи със набори BMP изображения.



## Фигура 1.5: VirtualDub

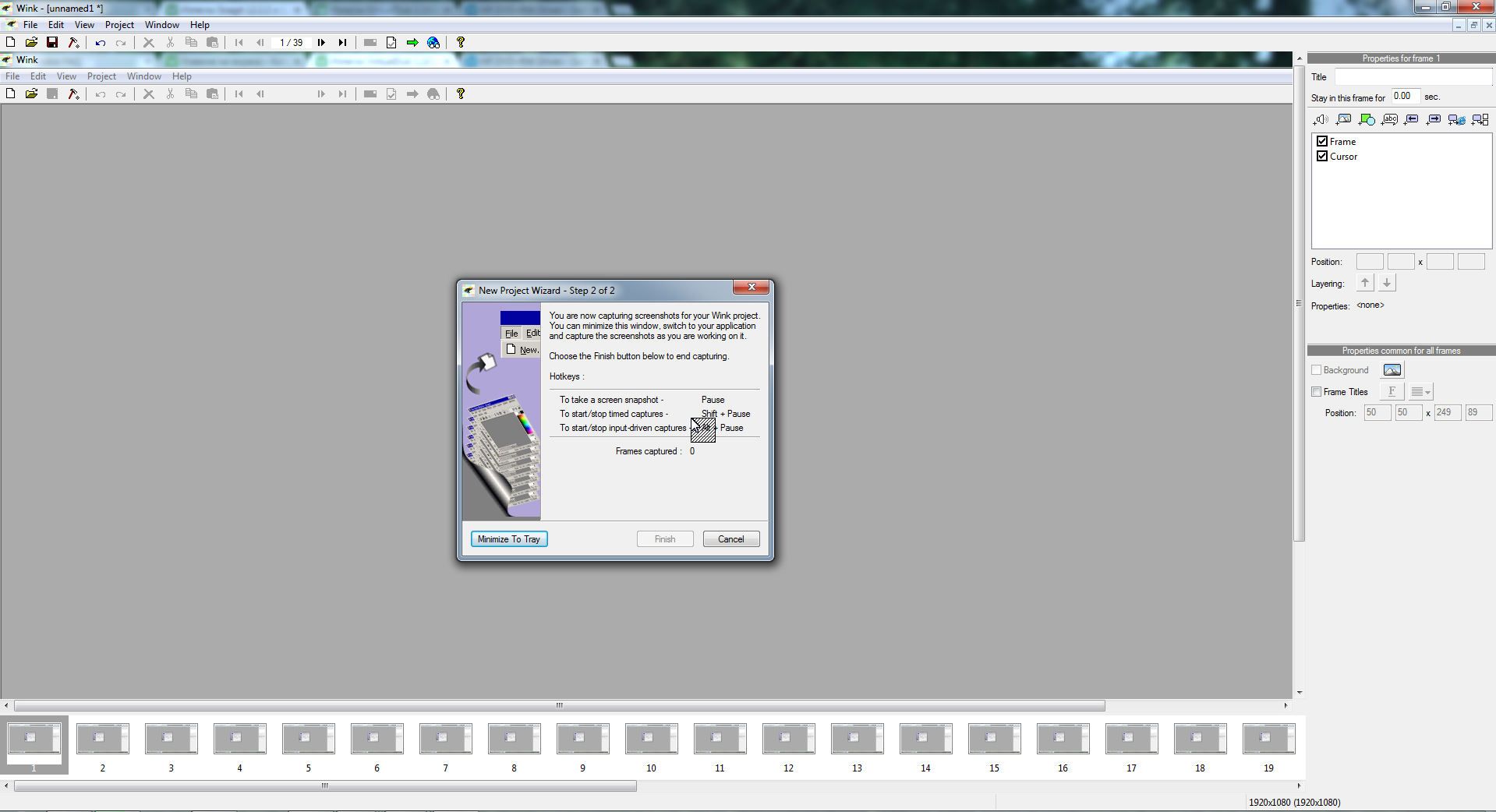
Програмата има 2 режима на работа: обработка и записване на екрана. В безплатната версия записването на екрана е изключено. В режим обработка предлага възможност за изрязване на част от едно видео, добавяне на филтри и запис в следните формати: avi, gif, tga, flm, bin [[12](http://www.virtualdub.org/)]

Wink

Wink е програма за създаване на обучителени и презентационни материали. Програмата дава възможност за записване на екрана като прави screenshot през равен интервал от време, добавяне на пояснителни текстови кутии, бутони, заглавия и др. Има функционалности за запис на звук. След приключване на заснемането има възможност за обработка на всеки индивидуален кадър.

Работи на десктоп платформа, като изисква Windows или Unix-базирана операционна система.

Предлага възможност за запис в HTML, pdf, PostScript, SWF, EXE [[13](http://www.debugmode.com/wink/)]



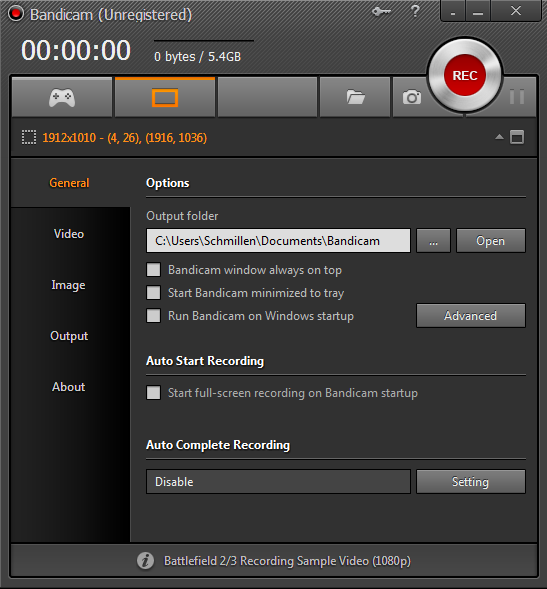
## Фигура 1.6: Wink

Bandicam

**Bandicam screen recorder** позволява записване на дадена област от екрана, или на програма която използва DirectX/OpenGL графични технологии. Предназначена е за заснемане на екран по време на играене на видео игри.

**Bandicam извършва видео компресиране от висока степен в формат avi или mp4. Има настройки за брой кадри в секунда, избор на регион от екрана за заснемане, избор на кодеци, ниво на качество и други.**

Предназначена е за десктоп платформи и изисква операционна система Windows. [[14](http://www.bandicam.com/)]



## Фигура 1.7: Bandicam

Sony Vegas Pro

Sony Vegas Pro е софтуер за видео обработка, разработвано от Sony Creative Software. Включва функционалности за многослойна видео и аудио обработка в реално време, преподреждане на набор от видеа с различни резолюции, сложни ефекти и композиционни инструменти, поддръжка на 24-bit/192 kHz аудио, поддръжка на VST и DirectX разширения и др. Работи с Windows (32-bit или 64-bit) операционни системи.

Програмата поддържа голям брой формати на видео, аудио и изображения: .aa3, .aif, .ac3, .flac, .bmp, .dpx, .exr, .jpeg, .png, .tiff, .wdp, .mp4, .mpg, .m2v, .m2t, .mpa, .mp3, .ogg, .rm, .avc, .mxf, .pca, .w64, .avi, .wav, .wma, .wmv и др. [[15](http://www.afterdawn.com/software/audio_video/video_editing/vegas_pro_32.cfm)]



## Фигура 1.8: Sony Vegas Pro

WeVideo

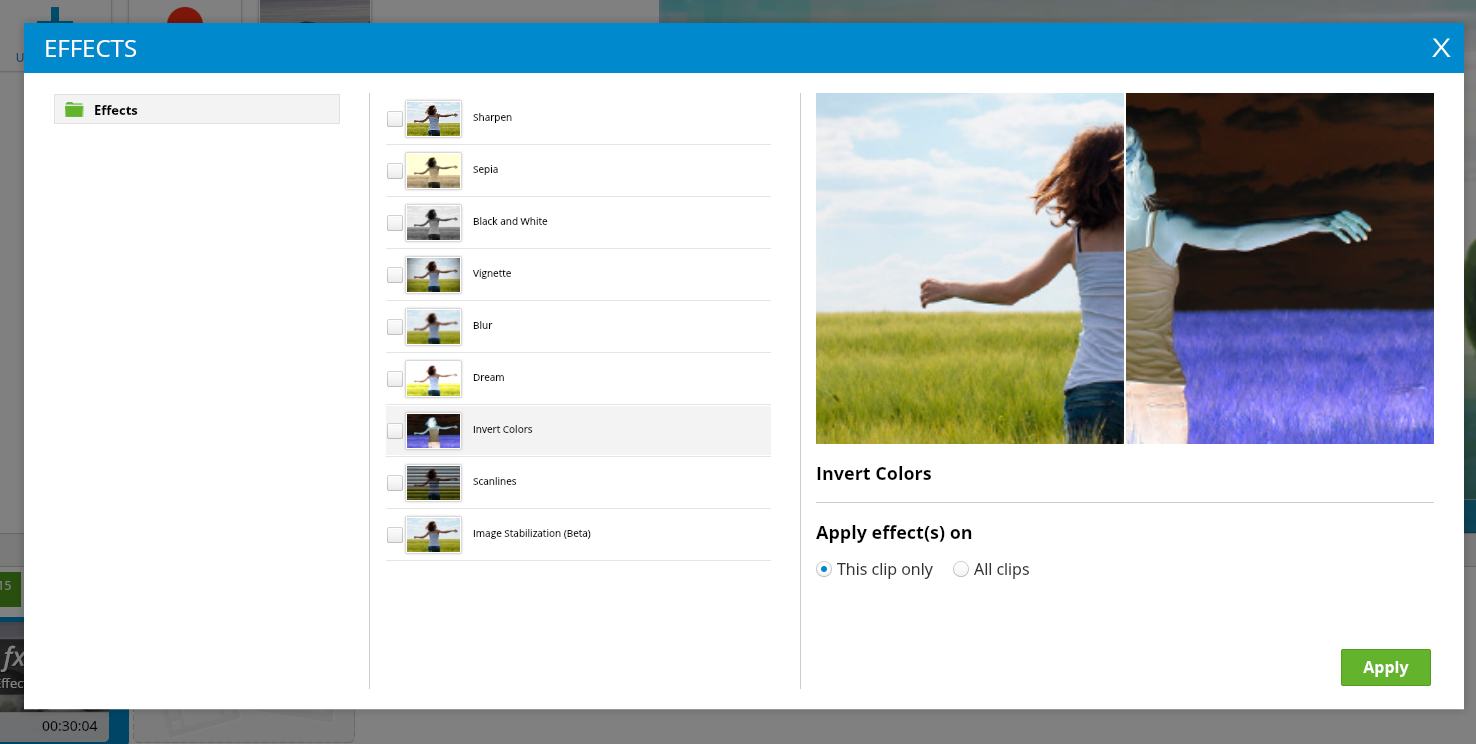
Платена онлайн уеб услуга, която предоставя видео обработка в браузъра. Предлагат няколко плана – free, premium, pro и други, като ограничават времетраенето на клиповете които потребителя има право да създава на месечна база.

От към функционалности за видео обработка има:

* Изрязване на клипове
* Избор на ефект от списък с предварително създадени такива
* Добавяне на текст
* Добавяне на аудио към видеото
* Записване на аудио

За да се направи обработка на видео, то трябва да бъде качено на сървъра. Потребителя има избор за качване от собствения си твърд диск, или от някой от облаците: google drive, dropbox, instagram, flickr, picasa, onedrive, box, facebook, dailymotion.

При публикуване на видеото потребителя избира качество (480p, 720p, 1080p, HD и други, в зависимост от вида на акаунта си) и има опцията или да го запише на своя твърд диск или да го качи на някоя друга уеб платформа (facebook, youtube и др.) [[16](https://www.wevideo.com/)]



## Фигура 1.9: Ефекти в WeVideo

# Технологична схема за обработка

Режим на идентификация/създаване потребител:

За достъпване на уебсайта се изисква потребителя да използва свой собствен акаунт. Системата обработва файлове, които са качени на сървъра. Всеки потребител има възможност да достъпва единствено своите файлове.

Началния екран (Р1) позволява създаването на нов потребителски акаунт или вписването във вече съществуващ такъв. Потребителя въвежда своите данни за идентификация – имейл и парола. При успешна идентификация се преминава към режима за зареждане на медийни файлове.

C:\Users\Schmillen\Desktop\ДОКУМЕНТАЦИЯ\mode1.png

## Фигура 2.1 Режим на идентификация/създаване на потребител

Режим на качване на зареждане на медийни файлове

След успешна идентификация потребителя е пренасочен към екрана за редакция (Р2).

От този екран може да се влезе в режим на зареждане на медийни файлове. Възможните типове файлове, които могат да бъдат заредени са: видео, аудио, изображения. За проверка проверка на типа на файловете се използва тяхната MIME информация.

Допустимите MIME типове за видео са:

* video/mp4
* video/quicktime
* video/x-msvideo
* video/x-ms-wmv

Допустимите MIME типове за аудио са:

* audio/mpeg
* audio/mp3
* audio/mp4
* audio/ogg
* audio/vnd.wav

Допустимите MIME типове за изображения са:

* image/jpeg
* image/png
* image/bmp

След избиране на файл от компютъра на потребителя, той бива качван на сървъра, където се поставя в папка, създадена за текущия потребител, до която единствено той има достъп. В базата данни се записва обща информация за файла – име, местоположение, тип, идентификационен номер на потребителя, както и мета данни, извлечени чрез библиотеката FFmpeg. В случай че файла е от тип видео, се генерира изображение –thumbnail – което представлява един кадър от средата на видеото и се визуално разпознаване на видеото в интерфейса на системата.

C:\Users\Schmillen\Desktop\ДОКУМЕНТАЦИЯ\mode2.png

## Фигура 2.2 Режим на качване на зареждане на медийни файлове

Режим на редактиране

В режим на редактиране се задават параметрите, които се използват за комбинирането и модифицирането на входните медийни файлове, чрез и създаването на финалното видео.

За избран входен медиен файл се задават начална и крайна точка от времевата линия и се прилагат различни филтри. След всяка промяна, направена от потребителя, данните от тези параметри се изпращат до сървъра в JSON формат през WebSocket, където биват използвани за генерирането на специфичната команда за запускане на FFmpeg процес по време на режим на рендиране.

C:\Users\Schmillen\Desktop\ДОКУМЕНТАЦИЯ\mode3.png

## Фигура 2.3 Режим на редактиране

Режим на преглед

След изпращане на параметрите на входните медийни файлове до сървъра, клиента прави http заявка за преглед на финалното видео. Сървъра използва параметрите, за да запусне FFmpeg процес, резултата от който бива пренасочен като поток от данни към клиента. В действителност в този момент се извършва рендиране, използвайки най-ниските настройки за качество, за да се осъществи възможно най-бързо рендиране. В зависимост дали се преглежда финалното видео или единично, резултатът се изобразява в съответния прозорец.

**C:\Users\Schmillen\Desktop\ДОКУМЕНТАЦИЯ\mode4.png**

*Фигура 2.4 Режим на преглед / рендиране*

Режим на рендиране

Режимът на рендиране е аналогичен на режима на преглед, с две разлики - потребителят задава опции за качество и формат и резултатът се записва в файл, а не се изпраща потоково към клиента. При приключване на рендирането се отваря нов прозорец, от където потребителят има възможност да изтегли финалното видео.

# Структура на програмната система

# Обща структура

Системата е изградена от няколко основни модули, връзките между които са изобразени в следната схема:

C:\Users\Schmillen\Desktop\ДОКУМЕНТАЦИЯ\diagram.png

## Фигура 3.1 Структурна схема на видео редактор

Модулите на системата могат да бъдат разделени в 2 основни категории – принадлежащи на сървърната част (бекенд), и принадлежащи на клиентската част (фронтенд).

Основният програмен език, с който е изградена системата, е javascript. Той е използван както за сървърната част, така и за функционалностите от клиентската част. За структурирането на уеб-страниците е използван HTML5, а за стилизирането им – CSS3.

За сървърната част е използван node.js, заедно с express.js фреймуърка. За съхранение на данните е избранa нерелационната база MongoDB, като библиотеката Mongoose е използвана за създаване на модели и за осъществяване на операциите с базата данни. Обработката на медийните файлове се извършва от FFmpeg, а Fluent-FFmpeg предоставя интерфейс за използване на FFmpeg в node.js среда.

Node.js е платформена среда за изпълнение на мрежови и сървърни приложения, които се пишат на javascript. Предоставя задвижвана от събития архитектура (event-driven) и неблокираща входно-изходна система за програмиране (API). Често се използва за приложения, работещи в реално време.

Npm е система за менажиране на пакети, която се използва по подразбиране с Node.js. Предоставя команден интерфейс за добавяне, премахване и обновяване на използваните библиотеки от дадено приложение.

Gulp.js е инструмент за автоматизиране на процеси по време на разработка. В конкретния случай се използва за стартиране на сървъра и следене за промяна в ключови файлове, след което автоматично се рестартира сървърния процес. Възможно е да се използва за минифициране на файлове, компресиране на изображения и др.

MongoDB е база данни от [нерелационен тип (NoSQL)](https://bg.wikipedia.org/wiki/NoSQL). Вместо да съхранява информация в таблици, както е при традиционните [релационни бази данни](https://bg.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8), MongoDB съхранява структурираната информация в [JSON](https://bg.wikipedia.org/wiki/JSON) формат с динамични схеми. Това прави интегрирането на информацията в определени приложения доста по-лесно и по-бързо. Когато се говори за нералационни бази данни, някои от основните термини трябва да бъдат заменени с техните аналогии в този контекст. Колекциите изпълняват ролята на таблиците, отделните записи се наричат документи, а колоните се наричат полета.

Mongoose е фреймуърк, който позволява лесно да се схематизират моделите на обектите. Документите в MongoDB са съставени от множество полета, които нямат строго определен тип. Това означава че по-подразбиране е възможно да се запише всякакъв тип информация в всяко поле, което е предпоставка за множество грешки. Mongoose осигурява логика на апликационно ниво, с която може да се ограничи типа на информация във всяко поле, и при опит за грешен запис да се хвърли грешка.

FFmpeg е безплатен продукт с отворен код, служещ за обработка на мултимедийна информация. Включва конзолен интерфейс. Кодът е разработен е главно на Linux, но може да бъде компилиран за други операционни системи – Mac OS X, Microsoft Windows, Android, iOS и др. Тази програма е в основната на разработения видео редактор. От нея се извършват всички мултимедийни модификации.

Fluent-ffmpeg е javascript библиотека, която служи за връзка между сървърната програма, написана на node.js и FFmpeg.

Клиентската част е съставена от 2 екрана – екран за вписване и регистриране, и екран за редактиране. Всички модули от клиентската част са разположени в екрана за редактиране. За изграждане на клиентската част са използвани библиотеките jQuery, jQuery.ui и фреймуърка bootstrap.css

HTML – Съкратено от Hyper Text Markup Language — на български "език за маркиране на хипертекст" е основният маркиращ език за описание и дизайн на уеб страници. HTML е стандарт в Интернет, а правилата се определят от международния консорциум W3C.

Описанието на документа става чрез специални елементи, наречени HTML елементи или маркери, които се състоят от етикети или тагове (HTML tags) и ъглови скоби (като например елемента <html>). HTML елементите са основната градивна единица на уеб страниците.

CSS (Cascading Style Sheets) е език за описание на стилове - използва се основно за описване на представянето на документ, написан на език за маркиране. Най-често се използва заедно с HTML, но може да се приложи върху произволен XML документ. Официално спецификацията на CSS се поддържа от W3C (World Wide Web Consortium). Създаден първоначално като средство за разделяне на съдържанието от представянето му, днес той се използва основно за визуално оформление на HTML страници.

CSS позволява да се определя как да изглеждат елементите на една HTML страница - шрифтове, размери, цветове, фонове, и др. CSS кодът се състои от последователност от стилови правила, всяко от които представлява селектор, последван от свойства и стойности.

Могат да се използват и разширения на CSS (Sass,Less) или фреймуърци(Foundation Zurb),които улесняват работата и позволяват допълнителна функционалност като наследяване,задаване на променливи и др.

CSS3 е последният CSS стандарт. Той дава възможност за множество нови начини за моделиране на страниците. Основната полза от CSS3 е, че той предлага множество нови опции за стилизиране на страниците, анимации, градиенти и в общи линии по-лесно изпълнение на по-сложни дизайни.

jQuery – jQuery е JavaScript фреймуърк, който изключително много улеснява програмирането с JavaScript. Освен това, гарантира, че приложенията ни ще работят под всички браузъри. jQuery ни позволява да създаваме ефекти, Ajax запитвания, модификация на DOM и др. jQuery предлага мощен инструментариум за селектиране на елементи в документ. Той работи с HTML таговете, техните идентификатори, класове, атрибути, имена и т.н. Може да се селектират елементи от цялото DOM дърво с отправна точка произволен елемент.

jQuery.ui е JavaScript библиотека, която предоставя колекция от фукционални графични интерфейсни елементи, включваща самодовършващи се текстови полета, плъзгачи, сортиращи се елементи, поле за избор на дата, помощни балончета и др. Представлява модифицируем jQuery плъгин, в който може да се включват само нужните функционалности за даден проект. По този начин размерът на библиотеката се ограничава, което намаля времето за зареждане на уеб страницата. В конкретния проект се използват елементите – draggable, droppable, slider, dialog и progressbar.

Bootstrap е безплатен пакет от инструменти оптимизирани за лесно създаване на УЕБ апликации. Той съдържа HTML и CSS базирани дизайн темплейти на графични елементи като бутони, форми, менюта, както и някои JavaScript разширения. Този фреймъурк дава възможност за стилизиране на приложения и освобождава УЕБ програмистите от мисълта за дизайн на интерфейса в процеса на разработка на прототипи.

Широко използван пакет, един от най-популярните в GitHub. Дори NASA го използват в своят УЕБ сайт.

Комуникацията между бекенда и фронтенда се осъществява чрез 2 протокола – HTTP и WebSocket.

HTTP (Hypertext Transfer Protocol) е мрежов протокол за пренос на хипертекст. Принадлежи на приложния слой на [OSI модела](https://bg.wikipedia.org/wiki/OSI_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB), служи за пренос на информация в [компютърни мрежи](https://bg.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8E%D1%82%D1%8A%D1%80%D0%BD%D0%B0_%D0%BC%D1%80%D0%B5%D0%B6%D0%B0). Създаден като средство за публикуване на [HTML](https://bg.wikipedia.org/wiki/HTML) страници, протоколът довежда до формирането на [Световната мрежа](https://bg.wikipedia.org/wiki/WWW). В HTTP протокола се използват понятия като [клиент](https://bg.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) и [сървър](https://bg.wikipedia.org/wiki/%D0%A1%D1%8A%D1%80%D0%B2%D1%8A%D1%80). HTTP е безсесиен протокол — това означава, че резултата на всяка следваща заявка не зависи от резултата на предишната и така всички клиенти получават равноправно еднакви ресурси. Съществуват различни способи за приложението на сесии в HTTP. Най-надеждният от тях е употребата на бисквитки (cookies).

WebSocket е протокол, позволяващ комуникация с пълен дуплекс през TCP връзка. Има възможност да бъде имплементиран в уеб-браузъри и уеб-сървъри, но също така може да се използва от всякакъв вид клиентски и сървърни приложения. Сходно на HTTP, при WebSocket се използва handshake метод за осъществяване на връзката. Позволява повече интеракция между браузъра и сървъра, като се използва трансфер на данни в реално време. По този начин се стандартизира начин сървъра да изпраща информация до клиента, без първоначална заявка от негова страна. Същото важи и в обратната посока. Комуникацията се извършва аналогично на HTTP протокола - през TCP, порт 80.

При зареждане на страницата за редактиране се отваря WebSocket връзка между клиента и сървъра, която се поддържа отворена докато потребителя използва системата.

Socket.IO е JavasScript библиотека за уеб приложения в реално време. Позволява двупосочна комуникация между уеб клиента и сървъра като използва WebSocket протокола. Съставена е от две части – клиентска част от библиотеката, която се зарежда в браузъра, и сървърна част от библиотеката, написана за Node.js. Двете части имат почти идентичен събитийно задвижван интерфейс.

# Структура на визуалния интерфейс на редактора

Визуалния интерфейс на редактора може да бъде разделен на няколко секции – Листинг на медийните файлове на потребителя (1), преглед на единична медия (2), преглед на финално видео (3) , формуляр за филтри (4) и времева лента (5). Елементите от тези секции се използват за задаване на параметрите на редакцията.

Фигура 3.2 Секции на визуалния интерфейс

Възможните манипулации, които могат да се правят върху медийните файлове са различни спрямо типа на файла, но в по-голямата си част се препокриват. За всички медийни файлове могат да се задават продължителност и начална точка от времевата лента. За звукови и видео файлове може да се регулират:

* Сила на звука – Определя стойността на филтъра “volume”. Допустимите стойности са в множеството [0;2] като 1 е по подразбиране. При 0 звукът е изключен, а при 2 е двойно по-силен от оригиналния.
* Обратно възпроизвеждане – Определя стойността на филтъра “reverse”. Може да бъде включено или изключено. При включен този филтър възпроизвежда медийния файл от зад напред. Често след включване на този филтър аудиото става трудно разбираемо.

За видео и изображения могат да се задават следните параметри:

* Контраст – Задава стойност на филтъра “contrast”. Допустимите стойности са между -2 и 2, като 1 е по подразбиране. При по-високи стойности се увеличава разликата в яркостта на различните области от картината.
* Яркост - Задава стойност на филтъра “brightness”. Допустимите стойности са между -1 и 1, като 0 е по подразбиране. При по-високи стойности се увеличава яркостта на цялата картина, а при по-ниски се затъмнява. При стойност -1 картината става черна.
* Наситеност - Задава стойност на филтъра “saturation”. Допустимите стойности са между 0 и 3, като 1 е по подразбиране. Определя наситеността на цветовете.
* Гама - Задава стойност на филтъра “gamma”, отговарящ за гама корекцията. Съществуват и отделни “gamma\_b”, “gamma\_r” и “gamma\_g” филтри, отговарящи за наситеноста на отделните цветове в изображението – съответно син, червен и зелен. Допустимите стойности са от 0.1 до 10, като 1 е по подразбиране.
* Наслагване на текст – Позволява поставянето на текст върху даден регион от картината. Възможно е да се регулира размер и цвят на текста, дали да бъде ограден от кутия и нейния цвят, както и координати за X и Y осите. Координатната система приема горния ляв ъгъл за отправна точка. При незададени координати, текста е изписан долу в средата на картината.
* Позиция на рамка – Задава параметри за “overlay” филтъра, който позволява да се определят височина и широчина, както и координати за медийния файл. Това позволява наслагване на няколко видея и/или изображения върху няколко слоя. Медиите върху по-горен слой ще бъдат на по-преден план при рендиране.

Възможно е да се създават празни медийни файлове (blank media), за които може да се избира цвят за фон, както и да се задава позиция на рамка и наслагване на текст.

В секция (1) са изобразени медийните файлове (клипове) на текущия потребител. От там той има възможност да добавя нови, като ги качва на сървъра от своя компютър или да изтрива вече съществуващи.

Клиповете, които ще бъдат включени във финалното видео се подреждат върху времевата лента (5). Там те могат да бъдат разтеглени или смалени, като по този начин се задава тяхната продължителност и начална точка. От секция 4 се задават стойностите на всички възможни филтри за текущо селектирания клип.

След записване на стойностите на филтрите или след промяна на клиповете върху времевата лента се обновява състоянието на редактора. В този момент се генерира JSON обект, съдържащ информацията за използваните медийни файлове и приложените върху тях филтри. Този обект се нарича обект на състоянието и се изпраща до сървъра през отворения WebSocket, където бива записан в сесийния обект.



## Фигура 3.3 Примерен обект на състоянието

Обновяването на обекта на състоянието се извърша по следния начин:

#### getEditorState: function ( single\_media ) {

#### ...

#### if ( !single\_media ) {

#### $layers = $('.layer');

#### var $media = [];

#### for ( var i = $layers.length - 1; i >= 0; i-- ) {

#### $layer\_media = $($layers[i]).find('.layer-media');

#### if ( $layer\_media.length ) {

#### for ( var j = 0; j < $layer\_media.length; j++ ) {

#### $media.push( $($layer\_media[j]) );

#### }

#### }

#### }

#### }

#### else {

#### $media = single\_media;

#### }

#### for( var i = 0; i < $media.length; i++) {

#### ...

#### timeline\_data.push({

#### 'file': $e.attr('data-filepath'),

#### 'start': start,

#### 'end': end,

#### 'offset': Number($e.attr('data-offset')),

#### 'timeline\_layer': 0,

#### 'volume': 1,

#### 'left': left,

#### 'top': top,

#### 'width': width,

#### 'height': height,

#### 'hasvideo': has\_video,

#### 'hasaudio': has\_audio,

#### 'filters' : filters

#### });

#### if ( end > duration ) {

#### duration = end;

#### }

#### }

#### $.editor.duration = duration;

#### return {'clips' : timeline\_data};

#### },

#### updateState: function( single\_media ) {

#### var state = this.getEditorState( single\_media );

#### this.socket.emit('update\_editor\_state', state);

#### if ( ! single\_media ) {

#### var video = document.getElementById('preview-final');

#### video.pause();

#### preview\_start = video.timelineTime;

#### video.src = "/preview";

#### if ( preview\_start ) {

#### video.src += "?preview\_start=" + preview\_start;

#### }

#### }

#### }

В секция (1) са изобразени всички медийни файлове, принадлежащи на текущия потребител. При кликване върху медиен файл, той бива селектиран, което вкарва приложението в режим на преглед на единична медия. По време на този режим „src“ атрибута на video тага, намиращ се в секция (2) придобива стойността на пътя до селектирания медиен файл, което позволява неговото изпълнение в браузъра.

#### updateInterfaceElements : function() {

#### ...

#### // Media element from the library is selected

#### if ( $selected\_media.hasClass('media') ) {

#### $('#delete-media').attr('disabled', false);

#### $('#preview-current').attr('src', $selected\_media.attr('data-filepath'));

#### $preview\_area.find('button, input, textarea').attr('disabled', true);

#### $preview\_area.find('.preview-control-button').attr('disabled', false);

#### $timeline.attr('data-offset', '0');

#### $timeline.attr('data-zoom', '1');

#### }

#### ...

#### }

При селектиране на клип, който вече е поставен върху времевата лента (5), е възможно задаването на стойности на отделните филтри. След задаването на поне 1 филтър прегледа на избраната медия изисква рендиране. “Src” атрибута на video тага от секция (2) се задава да бъде „/preview”, като обекта на състоянието се запълва единствено с данните за текущо избрания клип. Новата стойност на „src“ атрибута кара браузъра да направи HTTP заявка към указания url, от където се получава резултата от рендирането в webm формат.

Преглед на финалното видео се извършва по аналогичен начин, но вместо video елемента от (2), се използва този от (3). В обекта на състоянието се включват всички клипове от времевата лента.

# Структура на сървърна част

Тъй като JavaScript не е обектно-ориентиран програмен език, е трудно да се говори в този контекст за класове. Разделянето и капсулирането на функционалността се прави чрез модули. Основните модули на тази система са App, Routes, Editor.

C:\Users\Schmillen\Desktop\ДОКУМЕНТАЦИЯ\backend.png

## Фигура 3.4 Модули в бекенда

App – Този модул е ядрото на сървъра. В него се задават всички настройки, зареждат се използваните библиотеки, осъществява се връзката с базата данни и се активира „слушането“ за дадени събития.

Express е най-популярния фреймуърк за Node.js. Служи за повишаване нивото на абстракция на приложението и намаляване на количеството код, нужен за множество базови функционалности. Осигурява възможност за реализиране на MVC (model-view-controller) модел, но за целта е нужно използването на допълнителни модули.

Mustache Express – Модул, отговарящ за създаване на фронтенд темплейти. Това позволява при HTTP заявка за дадена страница, да се зареди съответното съдържание с правилните данни. Съставя View частта от MVC модела.

Mongoose – Отгоравя за изграждането на моделите от MVC структурата на приложението. Също осъществява връзката между приложението и базата данни и структуризира нерелационните данни в MongoDB.

Passport - Този модул е посредник за автентикация, който е изключително гъвкав и модулярен. Предоставя набор от „стратегии“ за автентикация по множество начини – чрез потребителско име и парола, Facebook или Twitter акаунт и множество други. В конкретния случай е използвана стратегията „LocalStrategy“, която изисква имейл и парола.

Session – Този модул предоставя начин за съхраняване на информация за потребителя. Тъй като HTTP протокола по принцип е безсесиен, приложението се нуждае от допълнителна такава функционалност.

Routes – Този модул играе ролята на диспечер на HTTP заявките. В него се описват действията, които да се извършват при получаване на заявка до всеки отделен “маршрут”, които изпълняват ролята на контролери от MVC модела. При създаване на маршрут се оказва URI, за който той отговаря, и метода на достъпване – GET, POST, PUT, DELETE. Реализираните маршрути са следните:

* router.get('/', function(req, res) { ... });

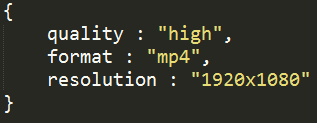
Изпълнява се при GET заявка до Root папката.

Отговаря за екрана за редакция. В случай че потребителя не е вписан в системата, го пренасочва към “/login”. Ако потребителя е вписан зарежда editor.html със данните от всички записи в Media документа, принадлежащи на потребителя.

* router.post('/editor', function(req, res) { ... });

Изпълнява се при POST заявка до “/editor”.

Отговаря за стартиране на рендиращия процес. Очаква да получи данни за настройка на рендирането в обекта render\_options със следния формат:



## Фигура 3.5 Примерен обект render\_options

Quality – приема 3 стойности: low, medium, high, който отговарят за качеството на звука. При low той се записва с sample rate от 32 kHz, при medium – 48 kHz, и при high – 96 kHz.

Генерираното видео се записва в потребителската папка. По време на рендиране изпраща информация за прогреса през отворения WebSocket по събитието „render-progress“, а при приключване изпраща пътя до файла по събитието „render-complete“.

При успешно изпълнение връща отговор 1, в противен случай връща отговор 0.

* router.get('/login', function(req, res){ ... });

Изпълнява се при GET заявка до “/login”.

Отговаря за екрана за вписване и регистрация. Зарежда login.html.

* router.post('/login' , function(req, res){ ... });

Изпълнява се при POST заявка до “/login”. Автентикира потребителя според изпратените данни за имейл и парола. При успешна автентикация препраща към “/”, от където се зарежда екрана за редакция.

* router.post('/delete-media', function(req, res){ ... });

Изпълнява се при POST заявка до “/delete-media”. Очаква да получи валиднен идентификационен номер на Медиен документ, който да бъде изтрит от базата данни. При успешно изтриване от базата, медийния файл се изтрива и от файловата система. При грешка връща отговор 0, a при успех 1.

* router.post('/register', function(req, res){ ... });

Изпълнява се при POST заявка до “/register”. Очаква да получи валиден имейл и 2 еднакви пароли. Създава потребител с зададения имейл и го вписва и пренасочва към “/”, където се зарежда екрана за редакция.

* router.post('/upload', function(req, res){ ... });

Изпълнява се при POST заявка до “/ upload ”. Отговаря за записване на медийни файлове на сървъра. Очаква да получи файловете в обекта req.files.files. Създава потребителска папка във файловата система с името на идентификационния номер на потребителя в папката public/media/ . В потребителската папка се създава папка за медийния файл. След завършване на качването се стартира процес ffprobe, който връща резултат с метаданни за медийния файл, които биват записани в базата данни.

var new\_media = new Media({ ... });

new\_media.save(function(err, newMedia) {

...

var command = ffmpeg(path)

.ffprobe(function(err, data){

...

Media.update({ \_id: newMedia.id }, {

$set: {

metadata: data,

filepath: new\_filepath

}

}, function(err, data){

...

if ( newMedia.type == 'video' ) {

(function(newMedia, folder){

var screenshot\_command = ffmpeg(new\_filepath);

screenshot\_command.screenshots({

count: 1,

timestamps: [duration / 2],

filename: 'thumb.jpg',

folder: folder,

size: '320x240',

}).on('end', function() {

Media.update({ \_id: newMedia.id }, {

$set: {

thumbnail: newMedia.getDirPath( false ) + 'thumb.jpg'

}

}, function(err, data){

...

console.log('Processing finished!');

req.app.io.emit('refresh');

});

});

})(newMedia, folder);

}

});

});

});

В случай, че файла е от тип видео, се генерира изображение от средния кадър, което да служи за визуално разпознаване на видеото. Изображението се записва като thumb.jpg в папката на новосъздадения медиен документ.

* router.get('/preview', function(req, res){ ... });

Изпълнява се при GET заявка до “/preview”. Стартира рендиране с данните от editor\_state обекта, записан в сесията, и следните настройки за качество:

#### { quality: “low”, format: “webm”, resolution: “320x180” }

Резултатът поточно се пренасочва response обекта, като по този начин видеото се изпраща към фронтенда, докато все още не е завършило рендирането. При наличен параметър “preview\_start” рендирането започва от означената секунда. Преди стартиране на новия процес за рендиране се проверява за вече съществуващ такъв от текущия потребител. При стартиране на рендиране модулът Editor връща като резултат FFmpeg javascript обект, който бива съхраняван в глобален масив с ключ, съответстващ на идентификационния номер на текущия потребител:

#### req.app.ffmpeg\_commands[req.user.\_id]

В случай, че такъв обект вече съществува, върху него се извиква метода kill(), който прекратява изпълнението на вече рендиращото се видео. Това предотвратява стартирането на множество процеси за рендиране в процеса на работа с графичния интерфейс.

#### res.contentType('webm');

#### var data = req.session.editor\_state;

#### if( data && req.user ){

#### if( req.app.ffmpeg\_commands[req.user.\_id] ) {

#### req.app.ffmpeg\_commands[req.user.\_id].kill();

#### req.app.ffmpeg\_commands[req.user.\_id] = false;

#### }

#### var ffmpeg\_preview = editor.render(data).format('webm');

#### var start = req.param("preview\_start");

#### if ( start && !isNaN(parseFloat(start)) && isFinite(start) ) {

#### ffmpeg\_preview.seekOutput(start);

#### }

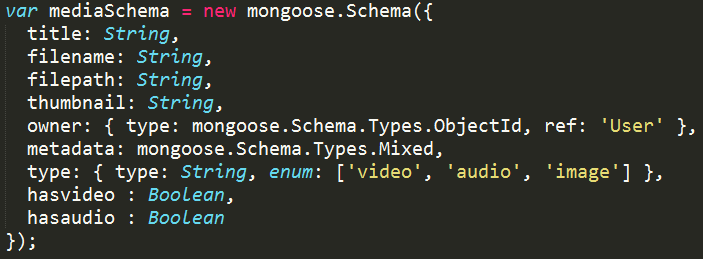
#### req.app.ffmpeg\_commands[req.user.\_id] = ffmpeg\_preview;

#### return ffmpeg\_preview.pipe(res, {end:true});

#### }

Editor – Този модул отговаря за генерирането на ffmpeg процес, който да рендира видеото по зададените параметри в обекта editor\_state, записан в сесията, и подадените стойности чрез render\_options. Разглежда се в дълбочина в точка 3.4

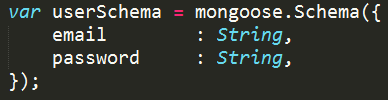
Media – Този модул представлява Mongoose модел, описващ данните в колекцията Medias, които отговарят за медийните файлове. Съдържа следната информация и структура:



## Фигура 3.6 Схематизиран модел на Медия

Полетата title, filename, filepath, type съдържат съответната на наименованието си информация. В полето thumbnail се записва пътя до създаденото в момента на качване изображение за визуално разпознаване. В “owner” се записва идентификационния номер на потребителя, създал медийния файл. В “metadata” се записва резултатът от командата ffprobe. “hasvideo” и “hasaudio” съдържа булева стойност, определяща дали медийния обект има видео и аудио потоци, които евентуално могат да подлежат на обработка.

User – Този модул представлява Mongoose модел, описващ данните в колекцията Users, които отговарят потребителите. Съдържа следната информация и структура.



## Фигура 3.7 Схематизиран модел на Потребител

В модула също се декларират методи за верификация на имейл и парола, както и за хеширане на парола. Зададено е правилото паролата да е с минимум 6 символа.

# Манипулация на медия с FFmpeg

Модула Editor отговаря за рендирането на видео по зададените от потребителя параметри, чрез визуалния интерфейс на видео редактора. За стартиране на процеса трябва да бъде извикана функцията render, която изглежда по следния начин:

#### render: function (data, options) {

#### ...

#### }

Като параметри приема информацията от editor\_state обекта, съхраняван в сесията, и настройки за качество на рендираното видео. При неналични настройки за качество по подразбиране се вземат следните стойности:

#### {

#### quality : "low",

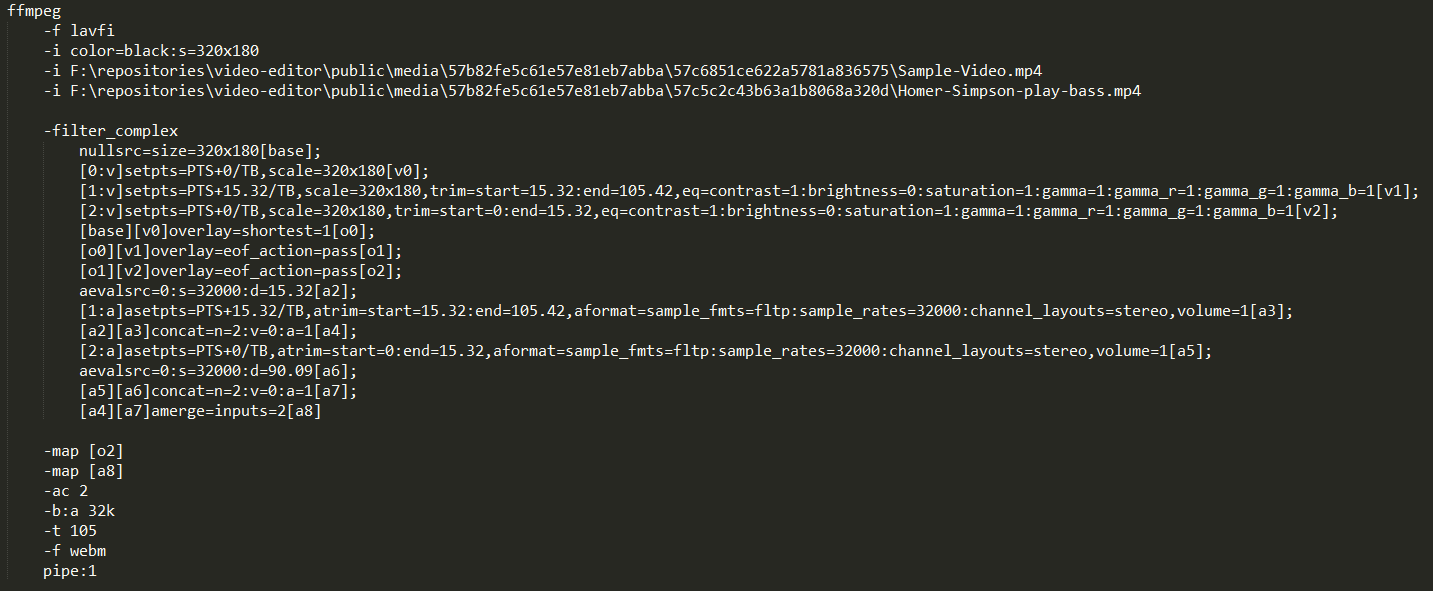
#### resolution : "320x180",

#### format : "webm"

#### }

За връзка между приложението и конзолната програма FFmpeg е използвана библиотеката fluent-ffmpeg.

Зададените параметри се използват за генерирането на команда, която се изпълнява като процес на сървърната машина. Една примерна команда има следния вид:



## Фигура 3.8 Примерна FFmpeg команда

Командата може да бъде разделена на 3 основни части – задаване на входни данни, настройки на филтрите, задаване на изходни данни. Всеки параметър има следния формат:

-име\_на\_параметър стойност\_на\_параметър

За входните данни може да се определи формат, но ако не е уточнен конкретен FFmpeg разпознава формата според файловото разширение.

Чрез квадратни скоби се селектират или именуват отделни потоци. Резултатното видео винаги има 1 видео и 1 аудио поток, като аудио потока е записан в 2 канала (Stereo).

Генерирането минава през следните стъпки:

* Създаване на ffmpeg обект чрез fluent-ffmpeg библиотеката и се декларират слушатели за събития:
* Задаване на входните файлове

За входни файлове се счита всеки обект от масива clips в editor\_state обекта. Ако даден файл присъства повече от веднъж в този масив, то той ще бъде добавен като съответен брой входни файла. Изпълнява се addInput метода върху ffmpeg обекта, и в масива на видео или аудио потоците се добавят генерираните опции.

#### ...

#### addInput: function(file, video\_options, audio\_options, format) {

#### this.ffmpeg.addInput(file);

#### if ( format ) {

#### this.ffmpeg.inputFormat(format);

#### }

#### if (video\_options) {

#### this.streams.video.push( this.addTrack( file, 'v', video\_options ) );

#### }

#### if (audio\_options) {

#### this.streams.audio.push( this.addTrack( file, 'a', audio\_options ) );

#### }

#### this.input\_index++;

#### }

#### ...

Първият input, който се добавя, е празен и се нарича базов. Има формат lavfi, който се използва за създаването на празен видео поток. Базовия input служи като най-долен слой на генерираното видео и чрез него се задава резолюцията на цялостното видео. Всички останали видео потоци се наслагват върху базовия.

#### // Add base input

#### this.addInput('color=' + background + ':s=' + resolution, {

#### declaration: [ 'setpts=PTS+0/TB', 'scale=' + resolution ],

#### overlay: [ 'overlay=shortest=1' ]

#### }, {}, 'lavfi');

* Задаване на филтри върху всеки входен файл

В тази стъпка има два вида филтри – такива които се задават по време на първоначална настройка, и такива които се задават впоследствие. За видео файловете втория вид филтри са свързани с Overlay, а при аудио файловете – с Concat

Филтрите при първоначална настройка служат за задаване на начално време за дадената медия, както и продължителност на нейната активност. В този момент също се определя времевото отместване спрямо началото на медийния файл. Възможно е времевия прозорец през който медията е активна, да не започва от нейното начало. Чрез филтъра “setpts” се задава момента в който медийния файл трябва да “тръгне”, с “trim” се определя частта от файла, която ще бъде изпълнена.

#### …

#### video\_options.declaration.push( 'setpts=PTS+' + (clip.start - offset) + '/TB' );

#### video\_options.declaration.push( 'scale=' + width + 'x' + height );

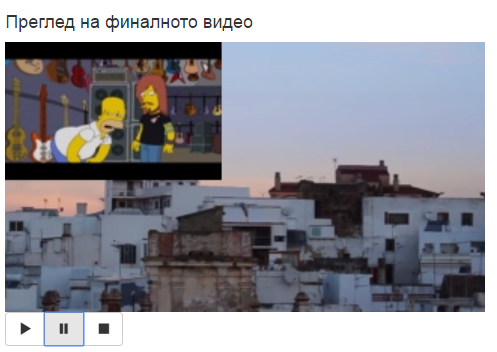
#### video\_options.declaration.push( 'trim=start=' + clip.start + ':end=' + (clip.end + offset) );

#### …

За видео файловете при първоначална настройка се задават стойности за всички филтри, които променят картината – brightness, contrast, saturation, gamma и др, както и размера на прозореца, в който ще бъде изобразено видеото.

За аудио файловете при първоначална настройка се задават стойности за sample\_rates, както и за сила на звука – volume.

Разработената система позволява наслагване на видеа едно върху друго. Това се осъществява чрез overlay филтъра.



## Фигура 3.9 Примерно наслагване на видеа

Построяването на overlay филтрите се осъществява последователно. Видео потока на всяко видео се наслагва върху изходния поток на предходното. Резултатния поток се подава на следващото видео. Последния изходен поток се пренасочва като видео поток на генерираното видео. На този етап също се задават координати на горния ляв ъгъл на насложеното видео.

#### // Create the video overlay step filter

#### var overlay\_filter = 'overlay=eof\_action=pass';

#### overlay\_is\_not\_full\_res = ( clip.top\_left\_x != 0

#### || clip.top\_left\_y != 0

#### || clip.bottom\_right\_x != resolution\_width

#### || clip.bottom\_right\_y != resolution\_height );

#### if ( overlay\_is\_not\_full\_res ) {

#### overlay\_filter += ":x=" + clip.top\_left\_x + ":y=" + clip.top\_left\_y;

#### }

#### video\_options.overlay.push( overlay\_filter );

#### …

#### var overlay\_index = 0;

#### for ( var i = 0; i < this.streams.video.length; i++ ) {

#### var stream = this.streams.video[i];

#### if ( stream.options.overlay && stream.options.overlay.length ) {

#### var filter = this.streamNameToString( this.streams.last\_video\_name ) + this.streamNameToString( stream.name );

#### filter += stream.options.overlay.join(',');

#### var overlay\_name = "o" + overlay\_index;

#### this.streams.last\_video\_name = overlay\_name;

#### filter += this.streamNameToString( overlay\_name );

#### overlay\_index ++;

#### this.addFilter( filter );

#### }

#### }

C:\Users\Schmillen\Desktop\ДОКУМЕНТАЦИЯ\overlays.png

## Фигура 3.10 Схема на наслагване

Всички аудио потоци се комбинират чрез филтъра “amerge”. За целта всички аудио потоци трябва да бъдат с еднакъв sample rate и с еднаква дължина. По време на декларация аудио потоците биват resample-нати на зададената стойност спрямо настройката за рендиране – quality. За задоволяване на второто условие всеки аудио поток бива конкатениран с 2 празни аудио потока – един от време 00:00 до началото на аудиото, и един от края на аудиото до края на рендирания файл.

C:\Users\Schmillen\Desktop\ДОКУМЕНТАЦИЯ\amerge.png

## Фигура 3.11 Схема на сливане на аудио потоци

* Пренасочване на филтрираните потоци към изходните потоци

Това се осъществява чрез параметъра map, който се добавя 2 пъти – първия за уточняване на видео потока, втория за аудио потока.

#### -map [o2]

#### -map [a8]

* Задаване на настройки за формат, продължителност и качество на звука

В тази стъпка се указват броя на аудио каналите, sample rate на звука, продължителност на цялото видео, както и формат.

* Уточняване на изход

Изходните потоци могат да бъдат записани във файл или да бъдат пренасочени по следния начин чрез задаване на следния изход - pipe:1

Всички стъпки без последната се извършват от Editor модула. Той създава javascript обект тип fluent-ffmpeg, извършва всички действия върху него и го връща на Routes модула. Там според текущия режим на работа – преглед или рендиране – се насочва изхода или към файл или към response обекта от Express.js

# Тестване на разработената програмна система

За тестване на системата бяха проведени следните тестове:

* White-Box Testing

Чрез този вид тестване се провери поведението на отделни модули от системата. Тестваха се различни функционални части. Анализирайки вътрешната структура на системата, се определиха в кои компоненти може да възникне дадено изключение. С избрани входни данни се извърши проверка дали на изхода се получи очаквания резултат. С този метод бяха открити няколко проблема и грешки, които можеха да попречат за правилното функциониране на цялостната система.

* Black-Box Testing

Функционалностите на системата бяха тествани без да се взема предвид вътрешната структура и функциониране. Извърши се чрез въвеждане на правилни и грешни входни данни за отделните модули, за да се определи коректността на изхода, но без да се обърне внимание на вътрешната структура.

* Unit Testing

Бяха извършени няколко Unit тестове, с които се установи коректността на данновите модели. Тестовете бяха извършени с цел отстраняване на проблеми във фазата на рефакториране на кода.

* System Testing

Системата беше тествана цялостно дали е в съответствие с функционалните изискванията и системната спецификация. Тук също не се взе предвид вътрешната структура на системата.

* Re-Testing

Приложени бяха и многократни тестове на функционалности с различни данни. С Retesting се установи дали при въвеждане на различни входни данни се получава очаквания краен резултат.

Резултати:

С проведените тестове на различните етапи от реализирането на системата бяха отстранени възникналите грешки и изключения. Успешно бяха тествани следните основни функционалности:

* Вписване в системата
* Регистриране на нов потребител
* Задаване на различни филтри на даден медиен обект
* Преглед на отделни медийни обекти
* Преглед на финалното видео
* Превъртане на финалното видео
* Рендиране

Получените резултати бяха в съответствие на изискванията, които бяха поставени при анализа на системата.

# Ръководство за потребителя

Ръководството има за цел да запознае потребителите на системата за първична обработка на видео с функционалните й възможности и да даде инструкции за работа, и да запознае сървърните администратори как да инсталират системата на нов сървър.

# Инсталация на нов сървър

Системата е разработена върху сървър с операционна система Windows, но също може да бъде инсталирана върху Unix базирани ОС. Видео редактора е разработен като проект с отворен код в платформата Github. Репозиторито се намира на следния адрес: <https://github.com/Ivo-Yankov/video-editor>

* Изтегляне на сървърните файлове:

Това може да се постигне по два начина - кода може да бъде изтеглен чрез командата

#### git clone [*git@github.com:Ivo-Yankov/video-editor.git*](mailto:git@github.com:Ivo-Yankov/video-editor.git)

Или може да бъде изтеглен като zip архив, който се разархивира в желаната папка.

* Инсталиране на FFmpeg

От този адрес изтеглете версията отговаряща на вашата операционна система и следвайте дадените инструкции:

<https://ffmpeg.org/download.html>

Ако операционната ви система е Windows трябва да създадете следните environment variables:

#### FC\_CONFIG\_DIR = Path\_to\_ffmpeg\conf

#### FONTCONFIG\_FILE= Path\_to\_ffmpeg\conf\fonts.conf

#### FONTCONFIG\_PATH = Path\_to\_ffmpeg\conf

В Path променливата трябва да се добави пътя до компилираните файлове на ffmpeg:

#### Path = …;Path\_to\_ffmpeg\bin

* Следвайте инструкциите за инсталиране на MongoDB от тук: <https://www.mongodb.com/download-center>

Стартирайте mongoDB процеса с командата:

#### mongod

* Следвайте инструкциите за инсталиране на Node.js за вашата операционна система от тук: <https://nodejs.org>
* Навигирайте до папката на проекта и изпълнете следните команди:

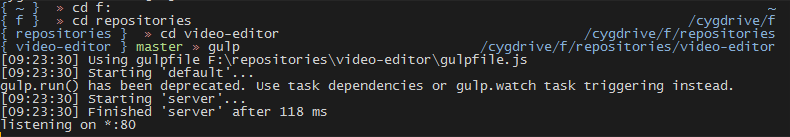
#### npm install

#### npm update

* Стартирайте сървъра с командата

#### gulp

При успешно стартиране ще получите следното съобщение и ще можете да достъпите сървъра на адрес <http://localhost:80>

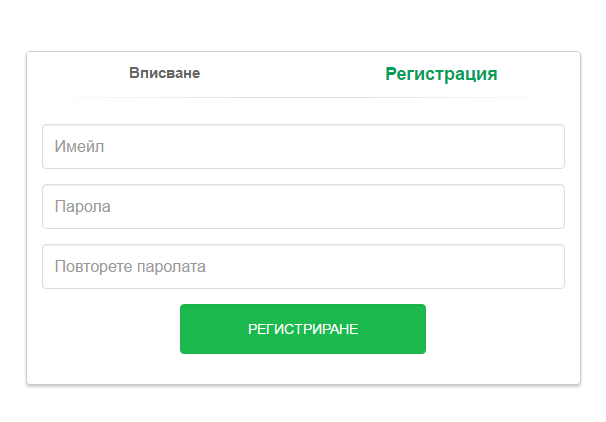


## Фигура 5.1 Успешно стартиран сървър

# Използване на системата

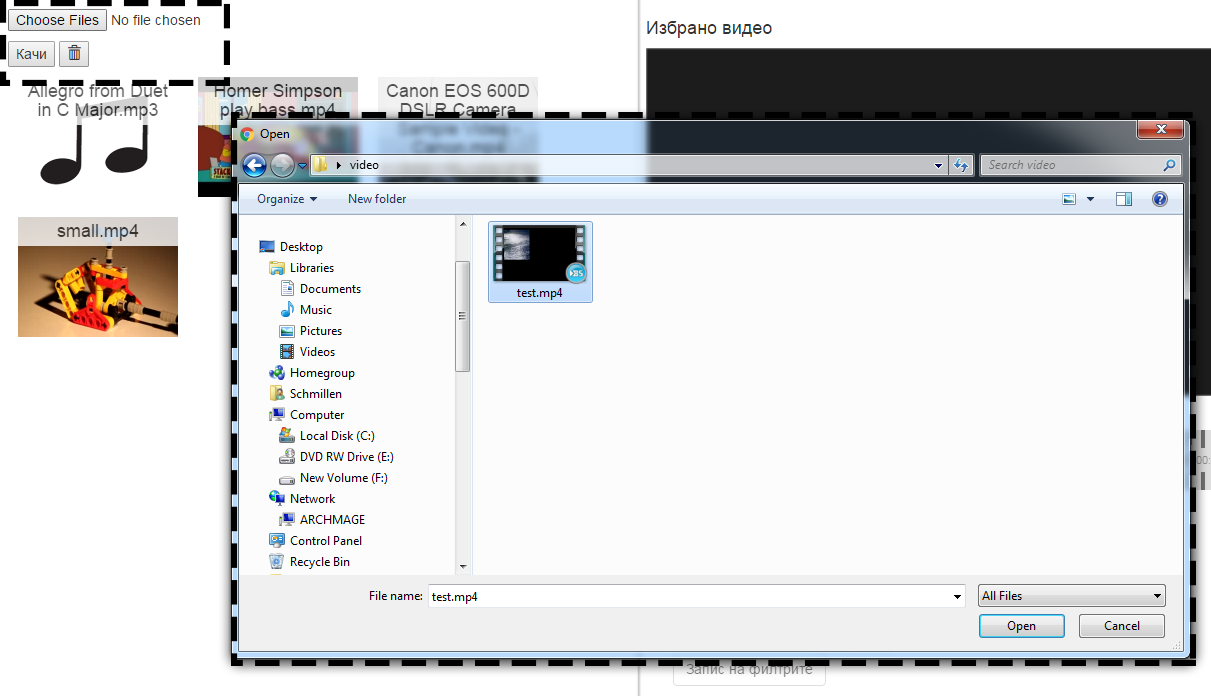
Видео редактора може да бъде достъпен през браузъра на този адрес: [*http://video.homeprojects.info/*](http://video.homeprojects.info/)

Първоначалната страница изисква потребителя да се впише в системата или да създаде нов потребителски акаунт. Трябва да бъдат въведени валиден имейл и парола, не по-кратка от 6 символа.



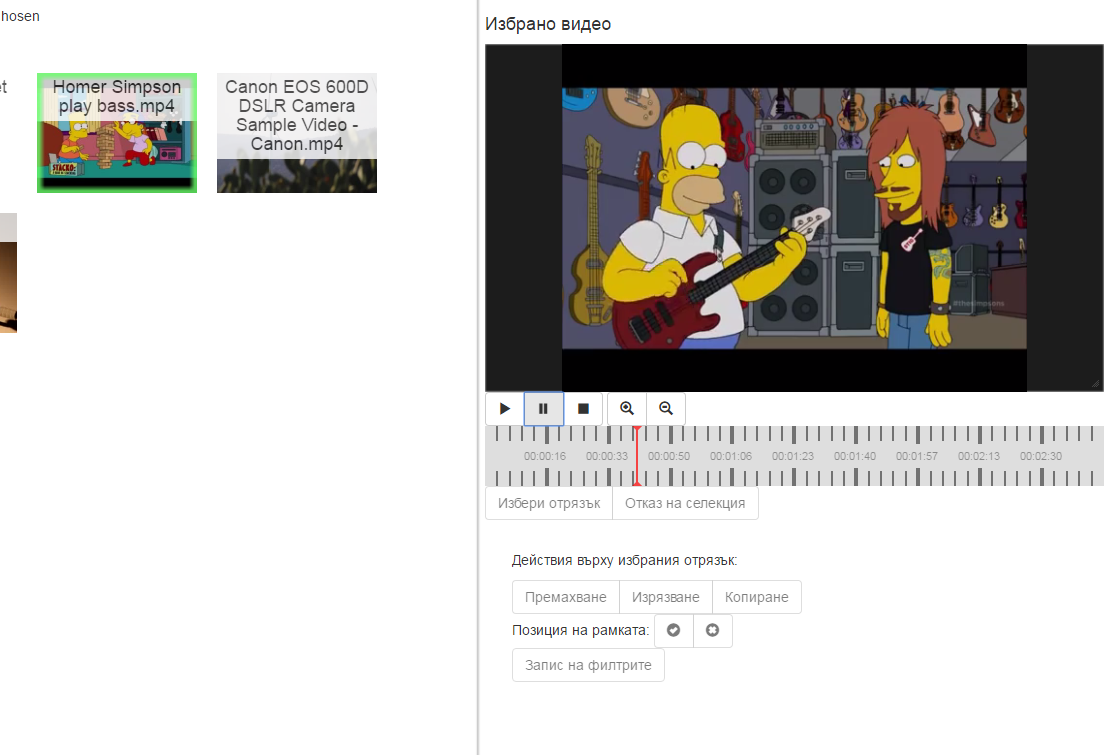
## Фигура 5.2 Екран за регистрация

След успешно вписване или регистрация потребителя е пренасочен към екрана за редакция. От там той има възможността да създаде видео, комбинирайки и променяйки свои медия файлове. За вкарване на нови медия файлове в системата се използва формата за upload:



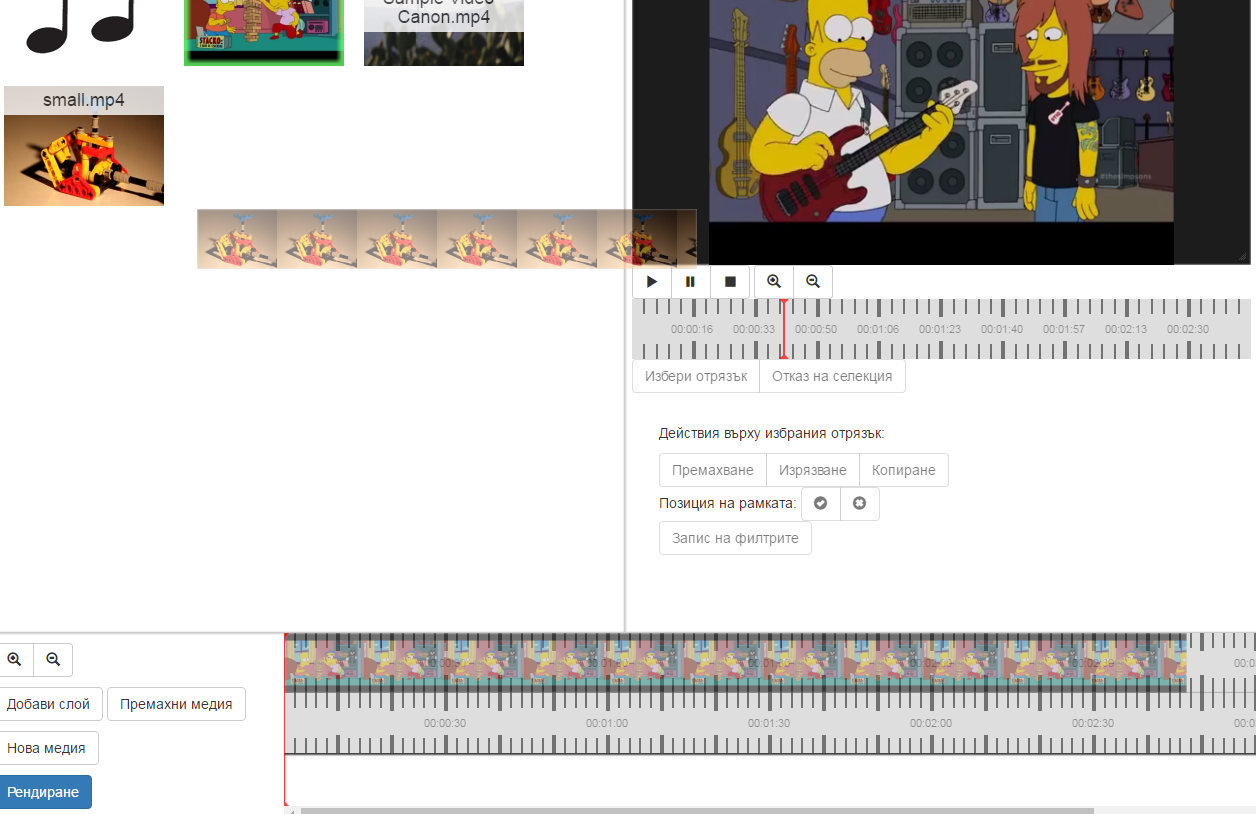
## Фигура 5.4: Качване на нови медия файлове

След успешно качване на медийните файлове може да се започне тяхната обработка. При единично кликване с ляв бутон върху медиен файл, той бива селектиран, което позволява неговия преглед в екрана „Избрано Видео“. Под видео екрана се намират бутони за стартиране, паузиране и спиране. Под тях се намира времевата лента на текущо селектираната медия. Червения маркер отбелязва сегашния момент. Възможно е влачене с мишката на маркера с цел превъртане напред или назад. Бутоните с лупи служат за приближаване и отдалечаване на времевата лента. При тяхното натискане се променя разстоянието между маркерите и стойността, която показват.



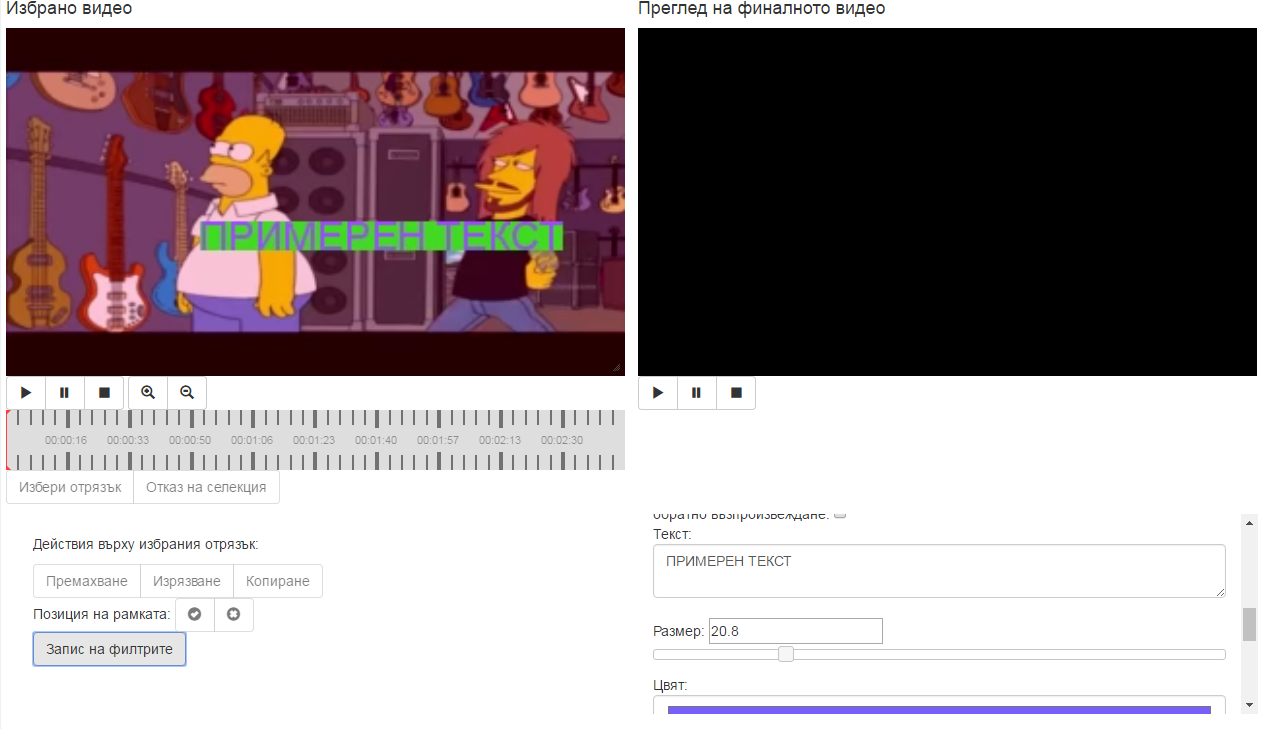
## Фигура 5.5: Преглед на селектирано видео

За да бъде редактирана някоя медия, тя трябва да бъде поставена върху времевата лента в дъното на екрана. Това се постига чрез влачене с ляв бутон на мишката. След като бъде поставена върху времевата лента, медията може да бъде влачена наляво и надясно, както и да бъде поставяна върху друг слой. Позицията на медията върху времевата лента регулира нейната начална и крайна точка във времето. При посочване с мишката в левия и десен край на медията се появяват ръкохватки, които се използват за оразмеряване. По този начин може медията да бъдат променени началния и краен момент на отделната медия.



## Фигура 5.6 Поставяне на медия върху времевата лента

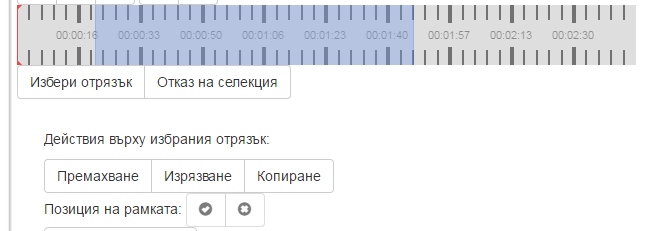
При единично натискане с ляв бутон върху медия, която се намира върху времевата лента, тя бива селектирана, което позволява нейното индивидуално редактиране. Екрана за добавяне на филтри се отключва. Според типа на меийния файл могат да се прилагат различни видове филтри върху медията. След задаване на стойностите трябва да се натисне бутона “Запис на филтрите”, след което резултата може да бъде прегледан както за индивидуалната медия, така и за финалното видео.



## Фигура 5.7 Примерни филтри

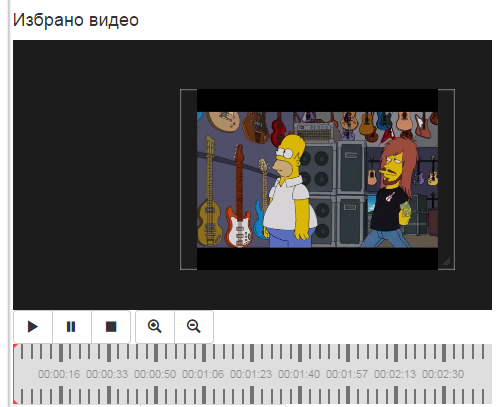
Докато е селектирана медия от времевата лента е възможно да се правят нейни изрязки. Това се постига чрез селектиране на даден участък от времевата лента на текущата медия и избиране на действие. Възможните действия са – премахване, изрязване, копиране. За избиране на времеви регион се натиска бутона “Избери Отрязък“, след което с ляв бутон на мишката върху времевата лента се поставя отрязъка, който може да бъде оразмеряван и позициониран. Бутона „Отказ на селекция“ премахва създадената селекция.

При избор на премахване, медията бива разделена на 2, като селектираната част се премахва. При избор на изрязване се оставя единствено селектирания регион, като останалата част се премахва. При копиране, се създава нова медия, която е идентична на селектираната, прилагат й се всички филтри, приложени на оригиналната и се изрязва избрания регион. Новата медия се поставя върху времевата лента веднага след селектираната.



## Фигура 5.8 Бутони за отрязъци

В долния десен ъгъл на екрана за преглед на избрано видео е разположена ръкохватка. При нейното плъзгане може да се оразмерява избраното видео и да бъде позиционирано някъде по екрана. Това позволява различни видеа да бъдат “насложени“ едно върху друго. След позициониране и оразмеряване трябва да се натисне положителния бутон „Позиция на рамката“. За връщане в първоначална позиция се натиска отрицателния бутон.

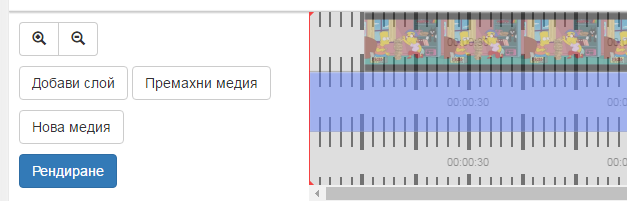


## Фигура 5.9 Позиция на рамката

В ляво от времевата лента са разположени бутони, за нейната манипулация. Бутоните с лупи са аналогични на тези, разположени под времевата лента за преглед на избрано видео. При натискане на “Добави Слой“ се добавя нов слой на времевата лента. Медийните обекти могат да бъдат поставяни върху всеки слой. При препокриване и при използване на ефекта “наслагване“ разположението по различните слоеве има значение. Обектите, които са в по-горен слой ще бъдат изобразени пред тези, разположени в по-долен.

Бутона „Премахни Медия“ премахва селектираната медия от времевата лента.

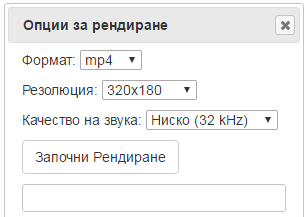
С „Нова Медия“ може да се добавя празен медиен обект. Тези обекти нямат видео или аудио информация, но върху тях могат да се поставят 2 филтъра – текстов и цвят на фон. Този тип медия може да се използва за фон на финалното видео.



## Фигура 5.10 Манипулация на времевата лента

При плъзгане на червения маркер се превърта финалното видео до желания момент за преглеждане.

Натискане на бутона “Рендиране“ отваря прозорец с опции за рендиране. Възможно е да се избере резолюция на финалното видео ( *320x180, 640x480, 800x600, 960x720, 1024x768, 1280x960, 1400x1050, 1440x1080, 1600x1200, 1856x1392, 1920x1080, 1920x1440* ), качество на звука ( *ниско – 32 kHz, средно – 48 kHz, високо – 96 kHz* ), както и формат на рендирания файл ( *mp4, webm, m4v, mpeg, mov, avi, wmv* ).



## Фигура 5.11 Опции за рендиране

С натискана на бутона „Започни Рендиране“ сървъра започва обработка на видеото. Прогресната лента под бутона се запълва в зависимост от процентната завършеност на рендирането. Когато обработката приключи се отваря нов прозорец, съдържащ резултатното видео, от където потребителя може да го изтегли с десен бутон на мишката или с клавишна комбинация crtl + s.

# Изводи и задачи на дипломната работа

Като основни цели на дипломната работа могат да се определят:

* Разработена е инструментална среда (система), която дава възможност за базова обработка на видео файлове. Има основни функционалности за изрязване и слепяне на клипове, добавяне на аудио, добавяне на презентационни елементи – текстове, илюстрации.
* Системата може да приема като входни данни множество различни видео формати и предлага разнообразие в форматите на изходните файлове.
* Системата е полезна за потребителите, защото е лесна за използване. Създаден е интуитивен интерфейс, с който потребителите лесно могат да се запознаят с основите на видео обработката.
* За допълнително улеснение на потрбителите системата е реализирана под формата на уебсайт, което елиминира нуждата от инсталиране на софтуер.

За бъдещо развитие могат да се реализират следните функционалности:

* Добавяне на по-голямо разнообразие от филтри
* Разработване на интерфейс, позволяващ осъществяването на ефект „зелен екран“.
* Добавяне на анимации
* Подобряване на комуникацията между клиента и сървъра – Преглед на видео да се осъществява чрез WebSocket протокола, а не чрез HTTP
* Възможност за създаване на аудио файлове
* Добавяне на възможност за запаметяване и зареждане на проект
* Отстраняване на новооткрити грешки и неточности

# Източници

1. Видео информация

https://en.wikipedia.org/wiki/Video

2. Кадри в секунда

https://en.wikipedia.org/wiki/Frame\_rate

3. Размерно съотношение на изображение

https://en.wikipedia.org/wiki/Aspect\_ratio\_(image)

4. Прогресивен формат на изобразяване

https://en.wikipedia.org/wiki/Progressive\_scan

5. Презредов формат на изобразяване

https://en.wikipedia.org/wiki/Interlaced\_video

6. Компресация на информация

https://en.wikipedia.org/wiki/Data\_compression

7. Кодеци

http://en.wikipedia.org/wiki/Codec

8. Обработка на видео информация

https://en.wikipedia.org/wiki/Video\_editing

9. Рендиране на видео

https://www.custcenter.com/app/answers/detail/a\_id/4506/~/what-does-render-mean%3F

10. VideoUniversity, Metadata For Video

http://www.videouniversity.com/articles/metadata-for-video/

11. Adobe

https://helpx.adobe.com/premiere-pro

12. VirtualDub

http://www.virtualdub.org/

13. DebugMode

http://www.debugmode.com/wink/

14. Bandicam

http://www.bandicam.com/

15. Sony Creative Software

http://www.sonycreativesoftware.com/vegaspro

16. WeVideo

https://www.wevideo.com

17. VesSoft

http://bg.vessoft.com/software/windows/download/bandicam

http://bg.vessoft.com/software/windows/download/wink

http://bg.vessoft.com/software/windows/download/virtualdub

# Приложение 1: Програмен код

App.js

#### var express = require('express');

#### var path = require('path');

#### var favicon = require('serve-favicon');

#### var logger = require('morgan');

#### var bodyParser = require('body-parser');

#### var mustacheExpress = require('mustache-express');

#### var passport = require('passport');

#### var flash = require('connect-flash'); //TODO

#### var routes = require('./routes/index');

#### var bb = require('express-busboy');

#### var session = require('express-session');

#### var sharedsession = require("express-socket.io-session");

#### var fs = require('fs');

#### var editor = require('./editor.js');

#### var app = express();

#### var http = require('http').Server(app);

#### var io = require('socket.io')(http);

#### app.io = io;

#### //database

#### var mongo = require('mongodb');

#### var mongoose = require('mongoose');

#### var configDB = require('./config/database.js');

#### mongoose.connect(configDB.url); // connect to our database

#### var db = mongoose.connection;

#### db.on('error', console.error.bind(console, 'connection error:'));

#### db.once('open', function (callback) {

#### // yay!

#### });

#### var configPassport = require('./config/passport'); // pass passport for configuration

#### var sessionMiddleWare = session({ secret: 'keyboard cat', resave: true, saveUninitialized: true });

#### app.use(sessionMiddleWare);

#### app.use(passport.initialize());

#### app.use(passport.session());

#### app.use(flash());

#### io.use(sharedsession(sessionMiddleWare));

#### app.use(function(req, res, next){

#### res.locals.success\_messages = req.flash('success\_messages');

#### res.locals.error\_messages = req.flash('error\_messages');

#### next();

#### });

#### passport.serializeUser(function(user, done) {

#### done(null, user);

#### });

#### passport.deserializeUser(function(user, done) {

#### done(null, user);

#### });

#### // view engine setup

#### app.engine('html', mustacheExpress());

#### app.set('view engine', 'mustache');

#### app.set('views', path.join(\_\_dirname, '/views'));

#### // uncomment after placing your favicon in /public

#### //app.use(favicon(path.join(\_\_dirname, 'public', 'favicon.ico')));

#### app.use(logger('dev'));

#### app.use(bodyParser.json());

#### app.use(bodyParser.urlencoded({ extended: true }));

#### app.use(express.static(\_\_dirname + '/public'));

#### bb.extend(app, { upload: true, path: 'public/media' });

#### app.use('/', routes);

#### app.\_router.stack.forEach(function(r){

#### if (r.route && r.route.path){

#### console.log(r.route.path)

#### }

#### })

#### app.ffmpeg\_commands = [];

#### io.on('connection', function(socket){

#### socket.on('update\_editor\_state', function(editor\_state){

#### socket.handshake.session.editor\_state = editor\_state;

#### socket.handshake.session.save();

#### var data = socket.handshake.session.editor\_state;

#### if( data ){

#### if( app.ffmpeg\_commands[socket.handshake.session.passport.user.\_id] ) {

#### app.ffmpeg\_commands[socket.handshake.session.passport.user.\_id].kill();

#### app.ffmpeg\_commands[socket.handshake.session.passport.user.\_id] = false;

#### }

#### }

#### else {

#### console.log('No editor state');

#### }

#### });

#### });

#### module.exports = app;

#### http.listen(80, function(){

#### console.log('listening on \*:80');

#### });

Routes/index.js

#### var express = require('express');

#### var router = express.Router();

#### var path = require('path');

#### var appDir = path.dirname(require.main.filename);

#### var mustacheExpress = require('mustache-express');

#### var fs = require('file-system');

#### var editor = require(path.join(appDir, '/editor.js'));

#### var bodyParser = require('body-parser');

#### var passport = require('passport');

#### var User = require("../models/user");

#### var Media = require("../models/media");

#### var ffmpeg = require('fluent-ffmpeg');

#### /\* GET home page. \*/

#### router.get('/', function(req, res) {

#### if ( !req.user ) {

#### return res.redirect('/login');

#### }

#### Media.find({}, function (err, medias) {

#### var visible\_medias = [];

#### for ( var i = 0; i < medias.length; i++ ) {

#### 

#### if (medias[i].type == 'audio') {

#### medias[i].thumbnail = 'images/audio-thumb.png';

#### }

#### else if (medias[i].type == 'image') {

#### medias[i].thumbnail = medias[i].filepath.replace('public', '');

#### }

#### if ( fs.existsSync(medias[i].filepath) ){

#### medias[i].filepath = medias[i].filepath.replace( 'public/', '' );

#### visible\_medias.push(medias[i]);

#### }

#### }

#### res.render('editor.html', {

#### medias: visible\_medias

#### });

#### });;

#### });

#### router.post('/editor', function(req, res) {

#### if (req.user) {

#### if ( !req.body.render\_options || !req.body.render\_options.format ) {

#### var format = 'webm';

#### }

#### else {

#### var format = req.body.render\_options.format;

#### }

#### 

#### var filename = "";

#### var possible\_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

#### for( var i=0; i < 10; i++ ) {

#### filename += possible\_chars.charAt(Math.floor(Math.random() \* possible\_chars.length));

#### }

#### var filepath = path.join('media/' + req.user.\_id, '/' + filename + '.' + format);

#### var data = editor.render(req.session.editor\_state, req.body.render\_options).on('progress', function(progress) {

#### req.app.io.emit('render-progress', progress);

#### })

#### .on('end', function() {

#### req.app.io.emit('render-complete', filepath);

#### })

#### .save( path.join(\_\_dirname, '../public/media/' + req.user.\_id, '/' + filename + '.' + format) );

#### res.send("1");

#### }

#### else {

#### res.send("0");

#### }

#### });

#### router.get('/login', function(req, res){

#### res.render('login.html', { success\_messages: req.flash('success\_messages'), error\_messages: req.flash('error\_messages') });

#### });

#### router.post('/delete-media', function(req, res){

#### if ( req.user ) {

#### Media.find({ \_id: req.body.id }, function (err, medias) {

#### if ( medias.length ) {

#### for ( var i = 0; i < medias.length; i++ ) {

#### var media = medias[i];

#### if ( media.owner == req.user.\_id ) {

#### media.remove( function (err) {

#### if (err) {

#### console.log(err);

#### res.send('0');

#### }

#### res.send('1');

#### });

#### }

#### else {

#### res.send('0');

#### }

#### }

#### }

#### else {

#### res.send('0');

#### }

#### });

#### }

#### else {

#### res.send('0');

#### }

#### });

#### router.post('/login', passport.authenticate( 'local', { successRedirect: '/', failureRedirect: '/login' } ));

#### router.post('/register', function (req, res) {

#### if ( !User.validateEmail(req.body.email) ) {

#### req.flash('error\_messages', 'Invalid email');

#### return res.redirect('/login');

#### }

#### if ( !User.validatePassword(req.body.password) ) {

#### req.flash('error\_messages', 'The password has to be at least 6 symbols long');

#### return res.redirect('/login');

#### }

#### var new\_user = new User({ email: req.body.email});

#### new\_user.password = new\_user.generateHash(req.body.password);

#### new\_user.save(function (err, user) {

#### if (err) console.error(err);

#### }).then(function( user ){

#### passport.authenticate('local', function(err, user) {

#### if (err || !user) {

#### return res.redirect('/login');

#### }

#### req.logIn(user, function(err) {

#### if (err) {

#### console.log(err);

#### return res.redirect('/login');

#### }

#### return res.redirect('/');

#### });

#### })(req, user);

#### });

#### });

#### router.post('/upload', function(req, res) {

#### var complete = function(err, file) {

#### if ( req.user ) {

#### var serverPath = file.filename;

#### var path = file.file;

#### var legit\_video\_mimes = [

#### 'video/mp4', // .mp4

#### 'video/quicktime', // .mov

#### 'video/x-msvideo', // .avi

#### 'video/x-ms-wmv' // .wmv

#### ];

#### var legit\_audio\_mimes = [

#### 'audio/mpeg', // mp3

#### 'audio/mp3', // mp3

#### 'audio/mp4', // mp4 audio

#### 'audio/ogg', // Ogg Vorbis

#### 'audio/vnd.wav' // wav

#### ];

#### var legit\_image\_mimes = [

#### 'image/jpeg',

#### 'image/png',

#### 'image/bmp',

#### ];

#### var file\_type = false;

#### var hasvideo = false;

#### var hasaudio = false;

#### if ( file.mimetype ) {

#### if ( legit\_video\_mimes.indexOf(file.mimetype) != -1 ) {

#### file\_type = "video";

#### hasaudio = true;

#### hasvideo = true;

#### }

#### if ( legit\_audio\_mimes.indexOf(file.mimetype) != -1 ) {

#### file\_type = "audio";

#### hasaudio = true;

#### }

#### if ( legit\_image\_mimes.indexOf(file.mimetype) != -1 ) {

#### file\_type = "image";

#### hasvideo = true;

#### }

#### }

#### if ( !err && file\_type !== false ) {

#### var new\_media = new Media({

#### title: serverPath,

#### filepath: path.replace('public',''),

#### filename: path.replace(/^.\*[\\\/]/, ''),

#### owner: req.user.\_id,

#### type: file\_type,

#### hasvideo: hasvideo,

#### hasaudio: hasaudio

#### });

#### new\_media.save(function(err, newMedia) {

#### if (err){

#### console.log(err);

#### }

#### //create a thumbnail

#### var command = ffmpeg(path)

#### .ffprobe(function(err, data){

#### if(err){

#### console.log(err);

#### }

#### var split\_filepath = data.format.filename.split('\\');

#### split\_filepath.pop();

#### var folder = newMedia.getDirPath(true);

#### 

#### var duration = data.format.duration;

#### if (!fs.existsSync(folder)){

#### fs.mkdirSync(folder);

#### }

#### var new\_filepath = folder + newMedia.filepath.replace(/^.\*[\\\/]/, '').replace(/\s/g, '-');

#### Media.update({ \_id: newMedia.id }, {

#### $set: {

#### metadata: data,

#### filepath: new\_filepath

#### }

#### }, function(err, data){

#### if(err){

#### console.log(err);

#### }

#### fs.renameSync(path, new\_filepath);

#### if ( newMedia.type != 'audio' ) {

#### (function(newMedia, folder){

#### var screenshot\_command = ffmpeg(new\_filepath);

#### screenshot\_command

#### .on('error', function(err){

#### console.log(err);

#### })

#### .screenshots({

#### count: 1,

#### timestamps: [duration / 2],

#### filename: 'thumb.jpg',

#### folder: folder,

#### size: '320x240',

#### }).on('end', function() {

#### 

#### 

#### Media.update({ \_id: newMedia.id }, {

#### $set: {

#### thumbnail: newMedia.getDirPath( false ) + 'thumb.jpg'

#### }

#### }, function(err, data){

#### if(err){

#### console.log(err);

#### }

#### console.log('Processing finished!');

#### req.app.io.emit('refresh');

#### });

#### });

#### })(newMedia, folder);

#### }

#### });

#### });

#### });

#### } else {

#### console.log(err);

#### }

#### }

#### }

#### if ( !req.files.files.length ) {

#### req.files.files = [req.files.files];

#### }

#### for ( var i=0; i<req.files.files.length; i++ ) {

#### var file = req.files.files[i];

#### (function(file){

#### var read = fs.createReadStream(file.file);

#### read.on('error', function(err) {

#### complete(err, file);

#### });

#### var write = fs.createWriteStream(file.filename);

#### write.on('error', function(err) {

#### complete(err, file);

#### });

#### write.on('close', function(err) {

#### complete(err, file);

#### });

#### read.pipe(write);

#### })(file);

#### }

#### res.redirect('/');

#### });

#### router.get('/preview', function(req, res) {

#### res.contentType('webm');

#### var data = req.session.editor\_state;

#### if( data && req.user ){

#### if( req.app.ffmpeg\_commands[req.user.\_id] ) {

#### req.app.ffmpeg\_commands[req.user.\_id].kill();

#### req.app.ffmpeg\_commands[req.user.\_id] = false;

#### }

#### var ffmpeg\_preview = editor.render(data).format('webm');

#### var start = req.param("preview\_start");

#### if ( start && !isNaN(parseFloat(start)) && isFinite(start) ) {

#### ffmpeg\_preview.seekOutput(start);

#### }

#### req.app.ffmpeg\_commands[req.user.\_id] = ffmpeg\_preview;

#### return ffmpeg\_preview.pipe(res, {end:true});

#### }

#### else {

#### console.log('No editor state');

#### return;

#### }

#### });

#### module.exports = router;

models/media.js

#### var mongoose = require("mongoose");

#### var mediaSchema = new mongoose.Schema({

#### title: String,

#### filename: String,

#### filepath: String,

#### thumbnail: String,

#### owner: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

#### metadata: mongoose.Schema.Types.Mixed,

#### type: { type: String, enum: ['video', 'audio', 'image'] },

#### hasvideo : Boolean,

#### hasaudio : Boolean

#### });

#### var Media = mongoose.model('mediacollection', mediaSchema);

#### mediaSchema.methods.getDirPath = function( include\_public) {

#### var public\_string = "";

#### if( include\_public ) {

#### public\_string = 'public/';

#### }

#### return public\_string + "media/" + this.owner + "/" + this.\_id + "/";

#### }

#### module.exports = mongoose.model('Media', mediaSchema);

models/user.js

#### var mongoose = require("mongoose");

#### var bcrypt = require('bcrypt-nodejs');

#### // define the schema for our user model

#### var userSchema = mongoose.Schema({

#### email : String,

#### password : String,

#### });

#### // methods ======================

#### // generating a hash

#### userSchema.methods.generateHash = function(password) {

#### return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null);

#### };

#### // // checking if password is valid

#### userSchema.methods.validPassword = function(password) {

#### return bcrypt.compareSync(password, this.password);

#### };

#### userSchema.statics.validateEmail = function(email) {

#### var re = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;

#### return re.test(email);

#### };

#### userSchema.statics.validatePassword = function(password) {

#### return password.length > 5;

#### };

#### // create the model for users and expose it to our app

#### module.exports = mongoose.model('User', userSchema);

config/database.js

#### // config/database.js

#### module.exports = {

#### 'url' : 'mongodb://localhost:27017/video\_editor'

#### };

Config/passport.js

#### var User = require('../models/user');

#### var passport = require('passport');

#### var LocalStrategy = require('passport-local').Strategy;

#### passport.use(new LocalStrategy({

#### usernameField: 'email',

#### passwordField: 'password'

#### },

#### function(email, password, done) {

#### User.findOne({ email: email }, function(err, user) {

#### if (err) {

#### return done(err);

#### }

#### if (!user) {

#### return done(null, false, { message: 'Incorrect email.' });

#### }

#### if (!user.validPassword(password)) {

#### return done(null, false, { message: 'Incorrect password.' });

#### }

#### return done(null, user);

#### });

#### }

#### ));

Editor.js

#### var fs = require('file-system');

#### var ffmpeg = require('fluent-ffmpeg');

#### var path = require('path');

#### var appDir = path.dirname(require.main.filename);

#### var clip = require(path.join(appDir, '/clip.js'));

#### module.exports = {

#### ffmpeg : {},

#### options : {},

#### streams : {

#### audio : [],

#### video : [],

#### last\_audio\_name : "a0",

#### last\_video\_name : "base"

#### },

#### complex\_filters : [],

#### input\_index : 0,

#### render: function (data, options) {

#### if ( !options ) {

#### options = {};

#### }

#### if ( !options.quality ) {

#### options.quality = "low";

#### }

#### if ( !options.resolution ) {

#### options.resolution = "320x180";

#### }

#### if ( !options.format ) {

#### options.format = "webm";

#### }

#### var audio\_samplerate;

#### switch ( options.quality ) {

#### case 'high' :

#### audio\_samplerate = "96000";

#### break;

#### case 'medium' :

#### audio\_samplerate = "48000";

#### break;

#### default:

#### audio\_samplerate = "32000";

#### break;

#### }

#### var clips = data.clips;

#### var resolution\_arr = options.resolution.split('x');

#### var resolution\_width = resolution\_arr[0];

#### var resolution\_height = resolution\_arr[1];

#### var resolution = options.resolution;

#### var background = "black";

#### var uploads\_folder = '/public/';

#### var eq\_filters = {

#### 'contrast' : 1,

#### 'brightness' : 0,

#### 'saturation' : 1,

#### 'gamma' : 1,

#### 'gamma\_r' : 1,

#### 'gamma\_g' : 1,

#### 'gamma\_b' : 1

#### };

#### this.init();

#### // Add base input

#### this.addInput('color=' + background + ':s=' + resolution, {

#### declaration: [ 'setpts=PTS+0/TB', 'scale=' + resolution ],

#### overlay: [ 'overlay=shortest=1' ]

#### }, {}, 'lavfi');

#### var duration = 0;

#### var there\_is\_video = false;

#### var there\_is\_audio = false;

#### for( var i = 0; i < clips.length; i++ ) {

#### var clip = clips[i];

#### clip.end = Number(clip.end);

#### clip.start = Number(clip.start);

#### var timeline\_end = clip.end;

#### if ( timeline\_end > duration ) {

#### duration = timeline\_end;

#### }

#### var filepath = path.join(\_\_dirname, uploads\_folder, clip.file);

#### var video\_options = {

#### declaration: [],

#### overlay : []

#### };

#### var audio\_options = {

#### start: clip.start,

#### end: clip.end,

#### filters: []

#### };

#### 

#### var offset = clip.offset;

#### if ( clip.hasvideo ) {

#### there\_is\_video = true;

#### // Convert the width, heigh, top and left properties from percentages to pixels

#### clip.top\_left\_x = parseInt(clip.left \* resolution\_width / 100);

#### clip.top\_left\_y = parseInt(clip.top \* resolution\_height / 100);

#### clip.bottom\_right\_x = parseInt(clip.top\_left\_x + clip.width \* resolution\_width / 100);

#### clip.bottom\_right\_y = parseInt(clip.top\_left\_y + clip.height \* resolution\_height / 100);

#### // Create the video declaration step filters

#### video\_options.declaration.push( 'setpts=PTS+' + (clip.start - offset) + '/TB' );

#### var width = clip.bottom\_right\_x - clip.top\_left\_x;

#### var height = clip.bottom\_right\_y - clip.top\_left\_y;

#### video\_options.declaration.push( 'scale=' + width + 'x' + height );

#### video\_options.declaration.push( 'trim=start=' + clip.start + ':end=' + (clip.end + offset) );

#### if( clip.filters ) {

#### var clip\_eq\_filters = [];

#### for( var filter in clip.filters ) {

#### if (clip.filters.hasOwnProperty(filter) ) {

#### if ( eq\_filters.hasOwnProperty(filter) ) {

#### clip\_eq\_filters.push( filter + "=" + clip.filters[filter] );

#### }

#### if ( filter === "text" ) {

#### var text\_filters = [];

#### if ( clip.filters[filter].text ) {

#### if ( !clip.filters[filter].x ) {

#### clip.filters[filter].x = "(w-tw)/2";

#### }

#### if ( !clip.filters[filter].y ) {

#### clip.filters[filter].y = "h-(2\*lh)";

#### }

#### for ( var text\_filter in clip.filters[filter] ) {

#### if ( clip.filters[filter].hasOwnProperty(text\_filter) && clip.filters[filter][text\_filter] ) {

#### text\_filters.push( text\_filter + '=' + clip.filters[filter][text\_filter] );

#### }

#### }

#### video\_options.declaration.push( "drawtext=" + text\_filters.join(":") );

#### }

#### }

#### }

#### }

#### if ( clip\_eq\_filters.length ) {

#### video\_options.declaration.push( "eq=" + clip\_eq\_filters.join(':') );

#### }

#### if ( clip.filters.reverse ) {

#### video\_options.declaration.push( "reverse" );

#### }

#### }

#### // Create the video overlay step filter

#### var overlay\_filter = 'overlay=eof\_action=pass';

#### overlay\_is\_not\_full\_res = ( clip.top\_left\_x != 0

#### || clip.top\_left\_y != 0

#### || clip.bottom\_right\_x != resolution\_width

#### || clip.bottom\_right\_y != resolution\_height );

#### if ( overlay\_is\_not\_full\_res ) {

#### overlay\_filter += ":x=" + clip.top\_left\_x + ":y=" + clip.top\_left\_y;

#### }

#### video\_options.overlay.push( overlay\_filter );

#### }

#### if ( clip.hasaudio ) {

#### there\_is\_audio = true;

#### var volume = 1;

#### if ( clip.filters && clip.filters.volume ) {

#### volume = clip.filters.volume;

#### }

#### // Create the audio filters

#### audio\_options.filters.push( 'asetpts=PTS+' + (clip.start - offset) + '/TB' );

#### audio\_options.filters.push( 'atrim=start=' + clip.start + ':end=' + (clip.end + offset) );

#### audio\_options.filters.push( 'aformat=sample\_fmts=fltp:sample\_rates=' + audio\_samplerate + ':channel\_layouts=stereo' );

#### if ( volume && volume !== 1 ) {

#### audio\_options.filters.push( 'volume=' + volume );

#### }

#### if ( clip.filters && clip.filters.reverse ) {

#### audio\_options.filters.push( 'areverse' );

#### }

#### }

#### 

#### // Add the input

#### if ( ! clip.file ) {

#### if ( clip.hasvideo && clip.filters.background\_color) {

#### this.addInput(

#### 'color=' + clip.filters.background\_color + ':s=' + width + 'x' + height,

#### video\_options,

#### false,

#### 'lavfi');

#### }

#### }

#### else {

#### this.addInput(filepath, video\_options, audio\_options);

#### }

#### }

#### //Create base video stream and set resolution

#### var baseFilter = 'nullsrc=';

#### baseFilter += 'size=' + resolution;

#### baseFilter += this.streamNameToString( this.streams.last\_video\_name );

#### this.addFilter( baseFilter );

#### //Declaration step

#### //Declare video streams

#### for ( var i = 0; i < this.streams.video.length; i++ ) {

#### var stream = this.streams.video[i];

#### if ( stream.options.declaration && stream.options.declaration.length ) {

#### // Get input stream

#### var filter = this.streamNameToString( stream.input\_index + ":v" );

#### // Apply declaration step options

#### filter += stream.options.declaration.join(',');

#### // Give this stream a name

#### filter += this.streamNameToString( stream.name );

#### this.addFilter( filter );

#### 

#### }

#### }

#### //Overlay step

#### var overlay\_index = 0;

#### for ( var i = 0; i < this.streams.video.length; i++ ) {

#### var stream = this.streams.video[i];

#### if ( stream.options.overlay && stream.options.overlay.length ) {

#### var filter = this.streamNameToString( this.streams.last\_video\_name ) + this.streamNameToString( stream.name );

#### // Apply overlay step options

#### filter += stream.options.overlay.join(',');

#### // Give a name to overlay

#### var overlay\_name = "o" + overlay\_index;

#### this.streams.last\_video\_name = overlay\_name;

#### //todo dont name the last overlay or map that stream to the output

#### filter += this.streamNameToString( overlay\_name );

#### //note: it is possible that

#### overlay\_index ++;

#### this.addFilter( filter );

#### }

#### }

#### //Audio step

#### var amerge\_filter = "";

#### var amerge\_count = 0;

#### for ( var i = 0; i < this.streams.audio.length; i++ ) {

#### filter = "";

#### this.incrementAudioStreamName();

#### concat\_filter\_start = "";

#### concat\_count = 1;

#### var stream = this.streams.audio[i];

#### if ( ! isEmpty( stream.options ) ) {

#### // Audio does not start at 0

#### if ( stream.options.start ) {

#### filter = 'aevalsrc=0:s=' + audio\_samplerate + ':d=' + stream.options.start + this.streamNameToString( this.streams.last\_audio\_name );

#### this.addFilter( filter );

#### concat\_filter\_start = this.streamNameToString( this.streams.last\_audio\_name );

#### this.incrementAudioStreamName();

#### concat\_count++;

#### }

#### filter = this.streamNameToString( stream.input\_index + ":a" );

#### filter += stream.options.filters.join(',');

#### 

#### var audio\_stream\_name = this.streamNameToString( this.streams.last\_audio\_name );

#### filter += audio\_stream\_name;

#### 

#### if ( ! ( stream.options.start || stream.options.end < duration ) ) {

#### amerge\_count++;

#### amerge\_filter += this.streamNameToString( this.streams.last\_audio\_name );

#### }

#### this.incrementAudioStreamName();

#### this.addFilter( filter );

#### // Audio does not end at the end of the whole movie

#### var concat\_filter\_end = "";

#### if ( stream.options.end < duration ) {

#### var end\_silence\_duration = duration - stream.options.end;

#### filter = 'aevalsrc=0:s=' + audio\_samplerate + ':d=' + end\_silence\_duration + this.streamNameToString( this.streams.last\_audio\_name );

#### this.addFilter( filter );

#### concat\_filter\_end = this.streamNameToString( this.streams.last\_audio\_name );

#### this.incrementAudioStreamName();

#### concat\_count++;

#### }

#### if ( stream.options.start || stream.options.end < duration ) {

#### var concat\_filter = concat\_filter\_start + audio\_stream\_name + concat\_filter\_end;

#### concat\_filter += "concat=n=" + concat\_count + ":v=0:a=1";

#### concat\_filter += this.streamNameToString( this.streams.last\_audio\_name );

#### this.addFilter( concat\_filter );

#### amerge\_count++;

#### amerge\_filter += this.streamNameToString( this.streams.last\_audio\_name );

#### // this.incrementAudioStreamName();

#### }

#### }

#### }

#### if ( amerge\_count > 1 ) {

#### this.incrementAudioStreamName();

#### amerge\_filter += "amerge=inputs=" + amerge\_count;

#### amerge\_filter += this.streamNameToString( this.streams.last\_audio\_name );

#### this.addFilter( amerge\_filter );

#### }

#### else if( !concat\_filter\_start && !concat\_filter\_end && this.streams.last\_audio\_name != 'a0' ) {

#### this.decrementAudioStreamName();

#### }

#### this.ffmpeg.complexFilter(this.complex\_filters);

#### // if (there\_is\_video) {

#### this.ffmpeg.addOption('-map', this.streamNameToString( this.streams.last\_video\_name ) );

#### // }

#### if (there\_is\_audio) {

#### this.ffmpeg.addOption('-map', this.streamNameToString( this.streams.last\_audio\_name ) );

#### this.ffmpeg.addOption('-ac', 2 );

#### this.ffmpeg.addOption('-b:a', (audio\_samplerate / 1000) + 'k' );

#### }

#### this.ffmpeg.addOption('-t', duration );

#### return this.ffmpeg;

#### },

#### init: function () {

#### this.ffmpeg = {};

#### this.options = {};

#### this.streams = {

#### audio : [],

#### video : [],

#### last\_audio\_name : "a0",

#### last\_video\_name : "base"

#### };

#### this.complex\_filters = [];

#### this.input\_index = 0;

#### this.ffmpeg = ffmpeg();

#### this.ffmpeg.on('progress', function(progress) {

#### console.log('Processing: ' + progress.timemark);

#### });

#### this.ffmpeg.on('error', function(err, stdout, stderr) {

#### console.log('Error: ' + err.message);

#### if ( stdout ) {

#### console.log("stdout:\n" + stdout);

#### }

#### if ( stderr ) {

#### console.log("stderr:\n" + stderr);

#### }

#### });

#### this.ffmpeg.on('end', function() {

#### console.log('Processing finished!');

#### });

#### this.ffmpeg.on('start', function(commandLine) {

#### console.log('Spawned Ffmpeg with command:');

#### console.log('---------');

#### console.log(commandLine);

#### console.log('---------');

#### });

#### },

#### setOutput: function(file){

#### this.ffmpeg.output(file);

#### },

#### addInput: function(file, video\_options, audio\_options, format) {

#### this.ffmpeg.addInput(file);

#### if ( format ) {

#### this.ffmpeg.inputFormat(format);

#### }

#### if (video\_options) {

#### this.streams.video.push( this.addTrack( file, 'v', video\_options ) );

#### }

#### 

#### if (audio\_options) {

#### this.streams.audio.push( this.addTrack( file, 'a', audio\_options ) );

#### }

#### this.input\_index++;

#### },

#### addTrack: function( file, type, options ) {

#### if( !options ){

#### options = [];

#### }

#### var input\_index = this.input\_index;

#### 

#### return {

#### filename : file,

#### name : type + "" + input\_index,

#### input\_index : input\_index,

#### options: options

#### };

#### },

#### setOption: function ( option, value) {

#### this.options[option] = value;

#### },

#### streamNameToString: function ( stream\_name ) {

#### return "[" + stream\_name + "]";

#### },

#### addFilter: function ( filter ) {

#### this.complex\_filters.push( filter );

#### },

#### incrementAudioStreamName: function() {

#### this.streams.last\_audio\_name = "a" + ( parseInt(this.streams.last\_audio\_name.replace("a", "")) + 1 );

#### },

#### decrementAudioStreamName: function() {

#### this.streams.last\_audio\_name = "a" + ( parseInt(this.streams.last\_audio\_name.replace("a", "")) - 1 );

#### },

#### };

#### function isEmpty(obj) {

#### for(var prop in obj) {

#### if(obj.hasOwnProperty(prop))

#### return false;

#### }

#### return true;

#### }

Views/login.html

#### <!DOCTYPE html>

#### <html>

#### <head>

#### <title>Video Editor</title>

#### <link rel="stylesheet" type="text/css" href="/stylesheets/jquery-ui.min.css">

#### <link rel="stylesheet" type="text/css" href="/stylesheets/jquery-ui.theme.min.css">

#### <link rel="stylesheet" type="text/css" href="/stylesheets/bootstrap.css">

#### <link rel="stylesheet" type="text/css" href="/stylesheets/login.css">

#### <script type="text/javascript" src="javascripts/jquery-3.1.0.min.js"></script>

#### <script type="text/javascript" src="javascripts/jquery-ui.min.js"></script>

#### <script>

#### $(function() {

#### $('#login-form-link').click(function(e) {

#### $("#login-form").delay(100).fadeIn(100);

#### $("#register-form").fadeOut(100);

#### $('#register-form-link').removeClass('active');

#### $(this).addClass('active');

#### e.preventDefault();

#### });

#### $('#register-form-link').click(function(e) {

#### $("#register-form").delay(100).fadeIn(100);

#### $("#login-form").fadeOut(100);

#### $('#login-form-link').removeClass('active');

#### $(this).addClass('active');

#### e.preventDefault();

#### });

#### });

#### </script>

#### </head>

#### <body>

#### </body>

#### </html>

#### <div class="container">

#### <div class="row">

#### <div class="col-md-6 col-md-offset-3">

#### <div class="panel panel-login">

#### <div class="panel-heading">

#### <div class="row">

#### <div class="col-xs-6">

#### <a href="#" class="active" id="login-form-link">Вписване</a>

#### </div>

#### <div class="col-xs-6">

#### <a href="#" id="register-form-link">Регистрация</a>

#### </div>

#### </div>

#### <hr>

#### </div>

#### <div class="panel-body">

#### <div class="row">

#### <div class="col-lg-12">

#### <form id="login-form" action="/login" method="post" role="form" style="display: block;">

#### <div class="form-group">

#### <input type="text" name="email" id="email" tabindex="1" class="form-control" placeholder="Имейл" value="">

#### </div>

#### <div class="form-group">

#### <input type="password" name="password" id="password" tabindex="2" class="form-control" placeholder="Парола">

#### </div>

#### <div class="form-group">

#### <div class="row">

#### <div class="col-sm-6 col-sm-offset-3">

#### <input type="submit" name="login-submit" id="login-submit" tabindex="4" class="form-control btn btn-login" value="Впиши се">

#### </div>

#### </div>

#### </div>

#### </form>

#### <form id="register-form" action="/register" method="post" role="form" style="display: none;">

#### <div class="form-group">

#### <input type="email" name="email" id="email" tabindex="1" class="form-control" placeholder="Имейл" value="">

#### </div>

#### <div class="form-group">

#### <input type="password" name="password" id="password" tabindex="2" class="form-control" placeholder="Парола">

#### </div>

#### <div class="form-group">

#### <input type="password" name="confirm-password" id="confirm-password" tabindex="2" class="form-control" placeholder="Повторете паролата">

#### </div>

#### <div class="form-group">

#### <div class="row">

#### <div class="col-sm-6 col-sm-offset-3">

#### <input type="submit" name="register-submit" id="register-submit" tabindex="4" class="form-control btn btn-register" value="Регистриране">

#### </div>

#### </div>

#### </div>

#### </form>

#### </div>

#### </div>

#### </div>

#### </div>

#### </div>

#### </div>

#### </div>

Views/editor.html

#### <!DOCTYPE html>

#### <html>

#### <head>

#### <title>Видео Редактор</title>

#### <link rel="stylesheet" type="text/css" href="/stylesheets/jquery-ui.min.css">

#### <link rel="stylesheet" type="text/css" href="/stylesheets/jquery-ui.theme.min.css">

#### <link rel="stylesheet" type="text/css" href="/stylesheets/bootstrap.css">

#### <link rel="stylesheet" type="text/css" href="/stylesheets/style.css">

#### <script type="text/javascript" src="javascripts/jquery-3.1.0.min.js"></script>

#### <script type="text/javascript" src="javascripts/jquery-ui.min.js"></script>

#### <script type="text/javascript" src="javascripts/socket.io.js"></script>

#### <script type="text/javascript" src="javascripts/scripts.js"></script>

#### </head>

#### <body>

#### </body>

#### </html>

#### <div id="media-area" class="ui-area">

#### <form ref='uploadForm'

#### id='uploadForm'

#### action='/upload'

#### method='POST'

#### encType="multipart/form-data">

#### <input type="file" name="files" multiple style="margin-bottom:10px;"/>

#### <input type='submit' value='Качи' /> <button id="delete-media"><span class="glyphicon glyphicon-trash"></span></button>

#### </form>

#### 

#### <div id="media-library" class="clearfix">

#### {{#medias}}

#### <div class="media" data-id="{{\_id}}" data-filepath="{{filepath}}" data-duration="{{metadata.format.duration}}" data-hasvideo={{hasvideo}} data-hasaudio={{hasaudio}} data-type={{type}} data-filters='{"text":{"text":"","fontsize":"16","fontcolor\_expr":"#000000","y":"","x":"","box":false,"boxcolor":"#000000"},"contrast":"1","brightness":"0","saturation":"1","gamma":"1","gamma\_r":"1","gamma\_g":"1","gamma\_b":"1","reverse":false,"volume":"1"}'>

#### <img src="{{thumbnail}}" draggable="false">

#### <h4 class="media-name">{{filename}}</h4>

#### </div>

#### {{/medias}}

#### </div>

#### </div>

#### <div id="preview-area" class="ui-area">

#### <div class="row">

#### <div class="preview-video" id="preview-current-area">

#### <h4>Избрано видео</h4>

#### <div class="preview-video-holder">

#### <div class="preview-video-handle">

#### <video id="preview-current" src=""></video>

#### </div>

#### </div>

#### 

#### <div class="btn-group" role="group">

#### <button class="btn btn-default preview-control-button" id="preview-current-play"><span class="glyphicon glyphicon-play"></span></button>

#### <button class="btn btn-default preview-control-button" id="preview-current-pause"><span class="glyphicon glyphicon-pause"></span></button>

#### <button class="btn btn-default preview-control-button" id="preview-current-stop"><span class="glyphicon glyphicon-stop"></span></button>

#### </div>

#### <div class="btn-group" role="group">

#### <button class="btn btn-default preview-control-button preview-current-zoom-button" data-amount="0.1" id="preview-current-zoom-in"><span class="glyphicon glyphicon-zoom-in"></span></button>

#### <button class="btn btn-default preview-control-button preview-current-zoom-button" data-amount="-0.1" id="preview-current-zoom-out"><span class="glyphicon glyphicon-zoom-out"></span></button>

#### </div>

#### <div class="timeline" data-data-zoom="1" data-markerdensity="0.5" data-duration="12" data-layerwidth="100" data-endtime="12"><div class="layer"></div></div>

#### <div class="btn-group" role="group">

#### <button class="btn btn-default" id="preview-current-selection">Избери отрязък</button>

#### <button class="btn btn-default" id="preview-current-remove-selections">Отказ на селекция</button>

#### </div>

#### </div>

#### <div class="preview-video">

#### <h4>Преглед на финалното видео</h4>

#### <div class="preview-video-holder">

#### <video id="preview-final" src=""></video>

#### </div>

#### <div class="btn-group" role="group">

#### <button class="btn btn-default preview-control-button" id="preview-final-play"><span class="glyphicon glyphicon-play"></span></button>

#### <button class="btn btn-default preview-control-button" id="preview-final-pause"><span class="glyphicon glyphicon-pause"></span></button>

#### <button class="btn btn-default preview-control-button" id="preview-final-stop"><span class="glyphicon glyphicon-stop"></span></button>

#### </div>

#### </div>

#### </div>

#### <div class="row filters">

#### <div class="col-xs-6">

#### <div style="margin: 20px 0 10px;">Действия върху избрания отрязък:</div>

#### <div class="btn-group" role="group">

#### <button class="btn btn-default" id="remove-selected">Премахване</button>

#### <button class="btn btn-default" id="remove-unselected">Изрязване</button>

#### <button class="btn btn-default" id="copy-media">Копиране</button>

#### </div>

#### 

#### <div>

#### Позиция на рамката:

#### <div class="btn-group" role="group">

#### <button class="btn btn-default" id="apply-overlay-position"><span class="glyphicon glyphicon-ok-sign"></span></button>

#### <button class="btn btn-default" id="reset-overlay-position"><span class="glyphicon glyphicon-remove-sign"></span></button>

#### </div>

#### </div>

#### <div>

#### <button class="btn btn-default" id="apply-fitlers">Запис на филтрите</button>

#### </div>

#### 

#### </div>

#### <div class="col-xs-6 filters-col">

#### <div class="audio-filters">

#### <div>

#### Сила на звука <input type="text" class="slider-input" readonly value="1" id="volume">

#### <div id="volume-slider" class="slider" data-min="0" data-max="2"></div>

#### </div>

#### </div>

#### <div class="video-filters">

#### <div>

#### Контраст <input class="slider-input eq-filter" type="text" readonly id="contrast" value="1"/>

#### <div class="slider" data-min="-2" data-max="2"></div>

#### </div>

#### <div>

#### Яркост

#### <input class="slider-input eq-filter" type="text" readonly id="brightness" value="0"/>

#### <div class="slider" data-min="-1" data-max="1"></div>

#### </div>

#### <div>

#### Наситеност <input class="slider-input eq-filter" type="text" readonly id="saturation" value="1" />

#### <div class="slider" data-min="0" data-max="3"></div>

#### </div>

#### <div>

#### Гама <input class="slider-input eq-filter" type="text" readonly id="gamma" value="1" />

#### <div class="slider" data-min="0.1" data-max="10"></div>

#### </div>

#### <div>

#### Гама (червено) <input class="slider-input eq-filter" type="text" readonly id="gamma\_r" value="1" />

#### <div class="slider" data-min="0.1" data-max="10"></div>

#### </div>

#### <div>

#### Гама (зелено) <input class="slider-input eq-filter" type="text" readonly id="gamma\_g" value="1" />

#### <div class="slider" data-min="0.1" data-max="10"></div>

#### </div>

#### <div>

#### Гама (синьо) <input class="slider-input eq-filter" type="text" readonly id="gamma\_b" value="1" />

#### <div class="slider" data-min="0.1" data-max="10"></div>

#### </div>

#### <div class="audio-filters">

#### обратно възпроизвеждане: <input type="checkbox" id="reverse-checkbox">

#### </div>

#### 

#### <div class="blank-filters video-filters">

#### Текст: <textarea class="form-control" id="media-text"></textarea><br>

#### <div>

#### Размер: <input class="slider-input" type="text" readonly id="media-text-size" value="16" />

#### <div class="slider" data-min="12" data-max="50"></div>

#### </div>

#### Цвят: <input class="form-control" type="color" id="media-text-color"><br>

#### Фон на текста: <input type="checkbox" id="media-text-box"><br>

#### Цвят на фона: <input class="form-control" type="color" id="media-text-boxcolor"><br>

#### Y координати (в пиксели): <input class="form-control" id="media-text-top"><br>

#### X координати (в пиксели): <input class="form-control" id="media-text-left"><br>

#### </div>

#### </div>

#### <div class="blank-filters">

#### Цвят на фон:

#### <input class="form-control" type="color" id="media-background-color">

#### </div>

#### </div>

#### </div>

#### </div>

#### <div id="timeline-area" class="ui-area">

#### <div id="timeline-controls" class="ui-area">

#### <div class="btn-group" role="group">

#### <button id="zoom-plus" class="btn btn-default zoom-button" data-amount="0.1"><span class="glyphicon glyphicon-zoom-in"></span></button>

#### <button id="zoom-minus" class="btn btn-default zoom-button" data-amount="-0.1"><span class="glyphicon glyphicon-zoom-out"></span></button>

#### </div>

#### <br>

#### <button class="btn btn-default" id="add-layer">Добави слой</button>

#### <button class="btn btn-default" id="remove-media">Премахни медия</button>

#### <button class="btn btn-default" id="add-media">Нова медия</button>

#### <br>

#### <button class="btn btn-primary" id="open-render-dialog">Рендиране</button>

#### </div>

#### <div class="ui-area timeline-holder">

#### <div id="timeline" class="timeline" data-zoom="1" data-markerdensity="1.5" data-duration="600" data-endtime="600" data-layerwidth="200">

#### <div class="layer"></div>

#### </div>

#### </div>

#### </div>

#### <div id="render-dialog" title="Render">

#### <form id="render-options" name="render-options">

#### Формат:

#### <select name="render-format" id="render-format">

#### <option value="mp4">mp4</option>

#### <option value="webm">webm</option>

#### <option value="m4v">m4v</option>

#### <option value="mpeg">mpeg</option>

#### <option value="mov">mov</option>

#### <option value="avi">avi</option>

#### <option value="wmv">wmv</option>

#### </select>

#### Резолюция:

#### <select name="render-resolution" id="render-resolution">

#### <option value="320x180">320x180</option>

#### <option value="640x480">640x480</option>

#### <option value="800x600">800x600</option>

#### <option value="960x720">960x720</option>

#### <option value="1024x768">1024x768</option>

#### <option value="1280x960">1280x960</option>

#### <option value="1400x1050">1400x1050</option>

#### <option value="1440x1080">1440x1080</option>

#### <option value="1600x1200">1600x1200</option>

#### <option value="1856x1392">1856x1392</option>

#### <option value="1920x1080">1920x1080</option>

#### <option value="1920x1440">1920x1440</option>

#### </select>

#### 

#### Качество на звука:

#### <select name="render-quality" id="render-quality">

#### <option value="low">Ниско (32 kHz)</option>

#### <option value="medium">Средно (48 kHz)</option>

#### <option value="high">Високо (96 kHz)</option>

#### </select>

#### <br>

#### <button class="btn btn-default" id="render">Започни Рендиране</button>

#### 

#### <div id="render-progress"></div>

#### </form>

#### </div>

Public/stylesheets/style.css

#### body {

#### padding: 0;

#### margin: 0;

#### height: 100vh;

#### overflow: hidden;

#### }

#### \*{

#### -webkit-touch-callout: none;

#### -webkit-user-select: none;

#### -khtml-user-select: none;

#### -moz-user-select: none;

#### -ms-user-select: none;

#### user-select: none;

#### }

#### .clearfix:after {

#### display: block;

#### content: "";

#### clear: both;

#### }

#### .ui-area {

#### -webkit-box-shadow: 1px 1px 1px 1px #ccc;

#### -moz-box-shadow: 1px 1px 1px 1px #ccc;

#### box-shadow: 1px 1px 1px 1px #ccc;

#### }

#### #media-area,

#### #preview-area {

#### height: 80%;

#### float: left;

#### padding: 10px;

#### }

#### #media-area {

#### width: 33.33%;

#### }

#### #preview-area {

#### width: 66.66%;

#### }

#### #timeline-area {

#### clear: both;

#### height: 20%;

#### border-top: 1px solid #ccc;

#### width: 100%;

#### }

#### .preview-video {

#### width: 48%;

#### float: left;

#### margin-left: 1%;

#### }

#### .preview-video-holder {

#### background: #1c1c1c;

#### overflow: hidden;

#### }

#### .preview-video-handle {

#### width: 100%;

#### height: 100%;

#### -webkit-box-shadow: inset 0px 0px 1px 0.5px rgba(225, 225, 225, 0.8);

#### -moz-box-shadow: inset 0px 0px 1px 0.5px rgba(225, 225, 225, 0.8);

#### box-shadow: inset 0px 0px 1px 0.5px rgba(225, 225, 225, 0.8);

#### }

#### .preview-video video {

#### width: 100%;

#### height: 100%;

#### }

#### .preview-video .timeline {

#### margin-left: 0;

#### height: auto;

#### }

#### #media-library {

#### max-height: 100%;

#### overflow: auto;

#### }

#### #media-library .media {

#### float: left;

#### width: 160px;

#### height: 120px;

#### position: relative;

#### text-align: center;

#### margin: 10px;

#### }

#### #media-library .media img {

#### position: absolute;

#### top: 0;

#### left: 0;

#### width: 100%;

#### height: 100%;

#### }

#### .media:after {

#### position: absolute;

#### top: 0;

#### left: 0;

#### height: 100%;

#### width: 100%;

#### content: "";

#### display: block;

#### }

#### #media-library .media-name {

#### background: white;

#### opacity: 0.8;

#### width: 100%;

#### position: absolute;

#### top: 0;

#### left: 0;

#### margin: 0;

#### padding: 5px;

#### }

#### .audio,

#### .layer-media[data-type=audio] {

#### background-color: #3cfff6;

#### }

#### .timeline-holder {

#### height: 100%;

#### }

#### #timeline-controls {

#### width: 300px;

#### float: left;

#### height: 100%;

#### padding: 10px;

#### }

#### #timeline-controls button{

#### margin-bottom: 10px;

#### }

#### .timeline {

#### margin-left: 300px;

#### overflow-x: auto;

#### height: 100%;

#### position: relative;

#### }

#### .timeline .layer {

#### width: 2000px;

#### height: 60px;

#### -webkit-box-shadow: 1px 1px 1px 1px #464545;

#### -moz-box-shadow: 1px 1px 1px 1px #464545;

#### box-shadow: 1px 1px 1px 1px #464545;

#### background: #dedede;

#### position: relative;

#### }

#### .timeline .layer:before {

#### position: absolute;

#### content: "";

#### display: block;

#### width: 0;

#### height: 100%;

#### top: 0;

#### left :0;

#### }

#### .timeline-ruler {

#### width: 100%;

#### position: absolute;

#### top: 0;

#### left: 0;

#### height: 100%;

#### font-size: 0.8em;

#### opacity: 0.7;

#### z-index:0;

#### overflow: hidden;

#### }

#### .ruler-marker {

#### width: 1px;

#### height: 30px;

#### line-height: 30px;

#### top: 50%;

#### margin-top: -15px;

#### position: absolute;

#### display: flex;

#### justify-content: center;

#### opacity: 0.7;

#### margin-left: -1px;

#### }

#### .ruler-marker:before,

#### .ruler-marker:after {

#### display: block;

#### content: "";

#### height: 20px;

#### width: 0px;

#### border: 1px solid black;

#### position: absolute;

#### left: 0;

#### }

#### .ruler-marker:before {

#### bottom: 100%;

#### }

#### .ruler-marker:after {

#### top: 100%;

#### }

#### .ruler-marker.big {

#### margin-left: -2px;

#### }

#### .ruler-marker.big:before,

#### .ruler-marker.big:after {

#### border: 2px solid black;

#### }

#### .ruler-marker.big:before {

#### bottom: 90%;

#### }

#### .ruler-marker.big:after {

#### top: 90%;

#### }

#### .layer-media {

#### height: 60px;

#### position: absolute !important;

#### left: 0;

#### top: 0;

#### width: 500px;

#### background-size: auto 100%;

#### z-index: 5;

#### opacity: 0.5;

#### -webkit-box-shadow: inset 0px 0px 1px 0.5px rgba(225, 225, 225, 0.8);

#### -moz-box-shadow: inset 0px 0px 1px 0.5px rgba(225, 225, 225, 0.8);

#### box-shadow: inset 0px 0px 1px 0.5px rgba(225, 225, 225, 0.8);

#### }

#### .layer-media.blank-media {

#### background-color: #6485ff;

#### }

#### .layer-media.selected,

#### .media.selected:after {

#### -webkit-box-shadow: inset 0px 0px 5px 5px rgba(105,255,105,0.8);

#### -moz-box-shadow: inset 0px 0px 5px 5px rgba(105,255,105,0.8);

#### box-shadow: inset 0px 0px 5px 5px rgba(105,255,105,0.8);

#### }

#### .time-marker {

#### position: absolute;

#### top: 0;

#### left: 0;

#### z-index: 100;

#### width: 10px;

#### height: 100%;

#### cursor: pointer;

#### opacity: 0.7;

#### margin-left: -5px;

#### }

#### .time-marker .line {

#### position: absolute;

#### top: 0;

#### left: 4px;

#### z-index: 100;

#### background: red;

#### width: 2px;

#### height: 100%;

#### }

#### .time-marker:after,

#### .time-marker:before {

#### position: absolute;

#### display: block;

#### z-index: 100;

#### left: 0;

#### content: "";

#### width: 0;

#### height: 0;

#### border-left: 5px solid transparent;

#### border-right: 5px solid transparent;

#### }

#### .time-marker:after{

#### bottom: 0;

#### border-bottom: 5px solid red;

#### }

#### .time-marker:before{

#### top: 0;

#### border-top: 5px solid red;

#### }

#### .timeline-selection {

#### position: absolute !important;

#### top: 0;

#### left: 0;

#### height: 100%;

#### width: 20px;

#### background: #7b9de8;

#### border-left: 1px solid #3b71ea;

#### border-right: 1px solid #3b71ea;

#### opacity: 0.4;

#### }

#### #render-dialog {

#### display: none;

#### }

#### .filters {

#### margin: 10px;

#### }

#### .slider {

#### margin: 5px 0 15px;

#### }

#### .row.filters {

#### height: 200px;

#### }

#### .filters-col {

#### max-height: 100%;

#### overflow: auto;

#### }

#### #render-options select,

#### #render-options button{

#### margin-bottom: 10px;

#### }

Public/javascripts/scripts.js

#### (function ( $ ) {

#### $.editor = {

#### pressed\_keys: [],

#### socket: "",

#### timeline: "",

#### preview\_timeline: "",

#### create\_selection: false,

#### preview\_video\_interval: "",

#### aspect\_ratio : {

#### width: 16,

#### height: 9

#### },

#### progressbar : "",

#### duration : 0,

#### init: function() {

#### this.socket = io();

#### this.socket.on('refresh', function() {

#### window.location.reload();

#### });

#### this.socket.on('render-complete', function( data ) {

#### window.open(data,'\_blank');

#### });

#### this.socket.on('render-progress', function( data ) {

#### // Convert hh:mm:ss to seconds

#### var timemark\_split = data.timemark.split(':');

#### var seconds = (+timemark\_split[0]) \* 60 \* 60 + (+timemark\_split[1]) \* 60 + (+timemark\_split[2]);

#### $.editor.progressbar.progressbar( 'value', seconds \* 100 / $.editor.duration );

#### });

#### this.timeline = new $.timeline( $('#timeline'), {

#### zoom\_button: $('.zoom-button'),

#### droppable: true,

#### time\_marker\_start: function(event, ui) {

#### var $timeline = $($.editor.timeline.timeline);

#### var $time\_marker = $timeline.find('.time-marker');

#### var video = document.getElementById('preview-final');

#### if ( !video.paused && !video.ended ) {

#### video.pause();

#### }

#### $time\_marker.stop();

#### },

#### time\_marker\_stop: function(event, ui) {

#### var $timeline = $($.editor.timeline.timeline);

#### var layer\_width = Number($timeline.find('.layer').width());

#### var end\_time = $timeline.attr('data-endtime');

#### var position\_percent = ui.position.left \* 100 / layer\_width;

#### var video = document.getElementById('preview-final');

#### document.getElementById('preview-final').timelineTime = position\_percent \* end\_time / 100;

#### $.editor.updateState();

#### },

#### } );

#### this.preview\_timeline = $.timeline( $('#preview-area .timeline'), {

#### zoom\_button: $('.preview-current-zoom-button'),

#### time\_marker\_drag: function(event, ui) {

#### var $timeline = $($.editor.preview\_timeline.timeline);

#### var $time\_marker = $timeline.find('.time-marker');

#### var layer\_width = Number($timeline.find('.layer').width());

#### var end\_time = $timeline.attr('data-endtime');

#### var position\_percent = ui.position.left \* 100 / layer\_width;

#### var video = document.getElementById('preview-current');

#### video.pause();

#### $time\_marker.stop();

#### document.getElementById('preview-current').currentTime = (position\_percent \* end\_time / 100) + Number($timeline.attr('data-offset'));

#### },

#### } );

#### document.getElementById('preview-current').ontimeupdate = function( e ) {

#### if ( this.stream ) {

#### var time\_delta = (this.currentTime - this.prevTime) || 0;

#### this.timelineTime += time\_delta;

#### $.editor.preview\_timeline.stopTimeMarker ( this.timelineTime );

#### this.prevTime = this.currentTime;

#### }

#### else {

#### var $timeline = $.editor.preview\_timeline.timeline;

#### var timeline\_offset = Number( $timeline.attr('data-offset') );

#### var timeline\_end = Number( $timeline.attr('data-duration') ) + timeline\_offset;

#### if ( timeline\_end && this.currentTime > timeline\_end ) {

#### this.pause();

#### this.currentTime = timeline\_end;

#### $.editor.preview\_timeline.stopTimeMarker( this.currentTime );

#### }

#### if ( timeline\_offset && this.currentTime < timeline\_offset ) {

#### this.currentTime = timeline\_offset;

#### }

#### }

#### };

#### var final\_video = document.getElementById('preview-final');

#### final\_video.timelineTime = 0;

#### final\_video.prevTime = 0

#### final\_video.ontimeupdate = function( e ) {

#### var time\_delta = (this.currentTime - this.prevTime) || 0;

#### this.timelineTime += time\_delta;

#### $.editor.timeline.stopTimeMarker ( this.timelineTime );

#### this.prevTime = this.currentTime;

#### };

#### $.editor.resizePreviewScreens();

#### $('.preview-video-handle').resizable({

#### handles: "e, w, n, s, ne, se, nw, sw"

#### }).draggable();

#### $('#preview-current-selection').on('click', function(){

#### $.editor.create\_selection = true;

#### });

#### $('#preview-current-remove-selections').on('click', function(){

#### $('#preview-current-area .timeline-selection').remove();

#### });

#### $('#apply-fitlers').on('click', function(){

#### $.editor.applyFilters();

#### });

#### this.preview\_timeline.timeline.on('click', function(e){

#### if ($.editor.create\_selection === true) {

#### var posX = $(this).offset().left;

#### var $timeline = $(this);

#### var $selection = $('<div class="timeline-selection"></div>');

#### 

#### $timeline.find('.timeline-selection').remove();

#### $selection.css('left', (e.pageX - posX) + "px");

#### $selection.resizable({

#### handles: "e, w"

#### });

#### $selection.draggable({

#### axis: "x",

#### containment: "parent",

#### scroll: true

#### });

#### $timeline.append($selection);

#### }

#### $.editor.create\_selection = false;

#### });

#### $('#preview-final-play').on('click', function() {

#### var video = document.getElementById('preview-final');

#### video.play();

#### $.editor.timeline.animateTimeMarker();

#### });

#### $('#preview-final-pause').on('click', function() {

#### var video = document.getElementById('preview-final');

#### video.pause();

#### $.editor.timeline.stopTimeMarker(video.timelineTime);

#### });

#### $('#preview-final-stop').on('click', function() {

#### var video = document.getElementById('preview-final');

#### video.timelineTime = 0;

#### video.prevTime = 0;

#### video.pause();

#### $.editor.timeline.stopTimeMarker(video.timelineTime);

#### $.editor.updateState();

#### });

#### $('#preview-current-play').on('click', function() {

#### var video = document.getElementById('preview-current');

#### video.play();

#### $.editor.preview\_timeline.animateTimeMarker();

#### });

#### $('#preview-current-pause').on('click', function() {

#### var video = document.getElementById('preview-current');

#### video.pause();

#### $.editor.preview\_timeline.stopTimeMarker(video.currentTime);

#### });

#### $('#preview-current-stop').on('click', function() {

#### var video = document.getElementById('preview-current');

#### video.currentTime = 0;

#### video.pause();

#### $.editor.preview\_timeline.stopTimeMarker( 0 );

#### if ( video.stream ) {

#### video.prevTime = 0;

#### video.timelineTime = 0;

#### $.editor.updateState();

#### }

#### });

#### $('#remove-selected').on('click', function() {

#### $.editor.cropMedia( 'remove-selected' );

#### });

#### $('#remove-unselected').on('click', function() {

#### $.editor.cropMedia( 'crop' );

#### });

#### $('#copy-media').on('click', function() {

#### $.editor.cropMedia( 'copy' );

#### });

#### $('#remove-media').on('click', function() {

#### $('.layer-media.selected').remove();

#### $.editor.updateState();

#### });

#### $('#add-media').on('click', function() {

#### $new\_layer = $.editor.timeline.addLayer();

#### $new\_media = $.editor.createLayerMedia({

#### style : '',

#### filepath : '',

#### start : 0,

#### duration : 10,

#### offset: 0,

#### filters : "",

#### type : "blank",

#### hasvideo : true,

#### hasaudio : false

#### })

#### 

#### $new\_media.addClass('blank-media');

#### $new\_layer.append($new\_media);

#### $.editor.initLayerMedia($new\_media);

#### $.editor.updateState();

#### });

#### $('#apply-overlay-position').on('click', function() {

#### var $selected\_media = $('.layer-media.selected');

#### $.editor.applyOverlayPosition( $selected\_media );

#### });

#### $('#reset-overlay-position').on('click', function() {

#### var $selected\_media = $('.layer-media.selected');

#### $.editor.resetOverlayPosition( $selected\_media );

#### });

#### $('#delete-media').on('click', function() {

#### var id = $('.media.selected').attr('data-id');

#### $.ajax({

#### url: "/delete-media",

#### data: {

#### id: id

#### },

#### method: "post",

#### dataType: "json"

#### }).done(function( data ) {

#### if (data == 1) {

#### $('.media[data-id=' + id + '], .layer-media[data-id=' + id + ']').remove();

#### }

#### });

#### return false;

#### });

#### $( ".media" )

#### .draggable({

#### snap: ".layer, .layer:before, .layer-media",

#### snapTolerance: 10,

#### helper: function() {

#### $this = $(this);

#### args = {

#### background : $this.find('img').attr('src'),

#### duration : $this.attr('data-duration'),

#### filepath : $this.attr('data-filepath'),

#### offset : 0,

#### hasvideo : $this.attr('data-hasvideo'),

#### hasaudio : $this.attr('data-hasaudio'),

#### type : $this.attr('data-type'),

#### filters : $this.attr('data-filters'),

#### }

#### return $.editor.createLayerMedia( args );

#### },

#### stop: function( event, ui ) {

#### ui.helper.remove();

#### }

#### })

#### .on('click', function() {

#### $('.selected').not(this).removeClass('selected');

#### $(this).toggleClass('selected');

#### $.editor.updateInterfaceElements();

#### });

#### $('#add-layer').on('click', function(){

#### $.editor.timeline.addLayer();

#### });

#### $('#open-render-dialog').on('click', function(){

#### $( "#render-dialog" ).dialog({title: "Опции за рендиране"});

#### });

#### $('#render').on('click', function(){

#### $( "#render-dialog" ).dialog();

#### var options = $.editor.renderVideo( {

#### quality : $('#render-quality').val(),

#### format : $('#render-format').val(),

#### resolution : $('#render-resolution').val()

#### } );

#### $.editor.updateState( options );

#### document.getElementById('preview-current').src = "";

#### document.getElementById('preview-final').src = "";

#### return false;

#### });

#### $('.slider').each( function(i, e) {

#### $e = $(e);

#### $e.slider({

#### min: Number($e.attr('data-min')),

#### max: Number($e.attr('data-max')),

#### step: 0.1,

#### value: $e.siblings('.slider-input').val(),

#### disabled: true,

#### slide: function( event, ui ) {

#### $(this).siblings('input').val( ui.value );

#### }

#### })

#### })

#### this.progressbar = $( "#render-progress" ).progressbar();

#### this.startHotKeyListener();

#### $.editor.updateInterfaceElements();

#### },

#### createLayerMedia : function ( args ) {

#### var data = "";

#### for (var key in args) {

#### if (args.hasOwnProperty(key) && key != 'background' && key != 'style') {

#### data += "data-" + key + "='" + args[key] + "' ";

#### }

#### }

#### var style = "";

#### if ( args.style ) {

#### style += args.style;

#### }

#### if ( args.background ) {

#### style += 'background-image:url("' + args.background + '")';

#### }

#### var $media = $("<div class='layer-media' " + data + " style='" + style + "'></div>");

#### return $media;

#### },

#### mediaOnClickHandler : function( e ){

#### $this = $(e.target);

#### 

#### $('.selected').not(e.target).removeClass('selected');

#### if ($this.hasClass('selected')) {

#### $this.removeClass('selected');

#### }

#### else {

#### $this.addClass('selected');

#### }

#### $.editor.updateInterfaceElements();

#### },

#### initLayerMedia : function ( media ) {

#### if ( !media ) {

#### media = $('.layer-media');

#### }

#### media.each(function(i,e) {

#### $e = $(e);

#### var start = Number($e.attr('data-start'));

#### var media\_duration = Number($e.attr('data-duration'));

#### var new\_left = ( start / $.editor.timeline.duration \* 100 );

#### $e.css('left', new\_left + "%" );

#### $e.css('width', ( media\_duration / $.editor.timeline.duration \* 100 ) + "%" );

#### if ($e.attr('data-initiated') !== 'true') {

#### $e.on('click', $.editor.mediaOnClickHandler);

#### 

#### $e.draggable({

#### snap: ".layer, .layer:before, .layer-media",

#### snapTolerance: 10

#### });

#### $e.resizable({

#### handles: "e, w",

#### stop: function( event, ui ) {

#### var $this = $(this);

#### var width = $this.width();

#### var timeline\_duration = Number($.editor.timeline.timeline.attr('data-duration'));

#### var timeline\_width = $this.closest('.layer').width();

#### $this.attr('data-duration', (timeline\_duration \* width / timeline\_width ) );

#### $.editor.updateState();

#### }

#### });

#### }

#### $e.attr('data-initiated', 'true');

#### });

#### },

#### startHotKeyListener: function( keys ){

#### $(document).on('keydown', function(e){

#### $.editor.pressed\_keys[e.keyCode] = true;

#### });

#### $(document).on('keyup', function(e){

#### $.editor.pressed\_keys[e.keyCode] = false;

#### });

#### },

#### isKeyPressed: function( key ) {

#### var keys = {

#### 'control': 17,

#### 'shift': 16

#### }

#### return Boolean(this.pressed\_keys[keys[key]]);

#### },

#### getEditorState: function ( single\_media ) {

#### var timeline\_data = [];

#### var start;

#### if ( !single\_media ) {

#### $layers = $('.layer');

#### var $media = [];

#### for ( var i = $layers.length - 1; i >= 0; i-- ) {

#### $layer\_media = $($layers[i]).find('.layer-media');

#### if ( $layer\_media.length ) {

#### for ( var j = 0; j < $layer\_media.length; j++ ) {

#### $media.push( $($layer\_media[j]) );

#### }

#### }

#### }

#### }

#### else {

#### $media = single\_media;

#### }

#### for( var i = 0; i < $media.length; i++) {

#### $e = $($media[i]);

#### if ( single\_media ) {

#### start = 0;

#### }

#### else {

#### start = Number($e.attr('data-start'));

#### }

#### var filters = $e.attr('data-filters');

#### if (filters && filters != 'undefined') {

#### filters = JSON.parse(filters);

#### }

#### var has\_video = $e.attr('data-hasvideo');

#### var has\_audio = $e.attr('data-hasaudio');

#### if ( !has\_video || has\_video == 'false') {

#### has\_video = false;

#### }

#### else {

#### has\_video = true;

#### }

#### if ( !has\_audio || has\_audio == 'false') {

#### has\_audio = false;

#### }

#### else {

#### has\_audio = true;

#### }

#### var top = 0;

#### var left = 0;

#### var width = 100;

#### var height = 100;

#### var overlay\_filters = $.editor.getFilter( $e, 'overlay' );

#### if ( overlay\_filters ) {

#### left = overlay\_filters.left || 0;

#### top = overlay\_filters.top || 0;

#### width = overlay\_filters.width || 100;

#### height = overlay\_filters.height || 100;

#### }

#### var duration = Number($e.attr('data-duration'));

#### var end = start + duration;

#### timeline\_data.push({

#### 'file': $e.attr('data-filepath'),

#### 'start': start,

#### 'end': end,

#### 'offset': Number($e.attr('data-offset')),

#### 'timeline\_layer': 0,

#### 'volume': 1,

#### 'left': left,

#### 'top': top,

#### 'width': width,

#### 'height': height,

#### 'hasvideo': has\_video,

#### 'hasaudio': has\_audio,

#### 'filters' : filters

#### });

#### if ( end > duration ) {

#### duration = end;

#### }

#### }

#### $.editor.duration = duration;

#### return {'clips' : timeline\_data};

#### },

#### renderVideo: function ( render\_options ) {

#### $.ajax({

#### url: "/editor",

#### data: {

#### render\_options: render\_options

#### },

#### method: "post",

#### dataType: "json"

#### }).done(function( data ) {

#### console.log(data);

#### });

#### },

#### updateState: function( single\_media ) {

#### var state = this.getEditorState( single\_media );

#### this.socket.emit('update\_editor\_state', state);

#### if ( ! single\_media ) {

#### var video = document.getElementById('preview-final');

#### video.pause();

#### preview\_start = video.timelineTime;

#### video.src = "/preview";

#### if ( preview\_start ) {

#### video.src += "?preview\_start=" + preview\_start;

#### }

#### }

#### },

#### cropMedia: function( action ) {

#### var $selected = $('.layer-media.selected');

#### var selection\_data = $.editor.preview\_timeline.getSelectionData();

#### var media\_duration = Number($selected.attr('data-duration'));

#### var media\_start = Number($selected.attr('data-start'));

#### var media\_end = media\_start + Number($selected.attr('data-duration'));

#### if (selection\_data.length) {

#### // Limit to 1 selection only

#### selection\_data = selection\_data[0];

#### }

#### else {

#### selection\_data.start = 0;

#### selection\_data.end = media\_duration;

#### }

#### var media\_arr = [];

#### if ( selection\_data.end > media\_duration ) {

#### selection\_data.end = media\_duration;

#### }

#### if ( selection\_data.start > media\_duration ) {

#### selection\_data.start = media\_duration;

#### }

#### if ( selection\_data.start < 0 ) {

#### selection\_data.start = 0;

#### }

#### if ( selection\_data.end < 0 ) {

#### selection\_data.end = 0;

#### }

#### var media\_offset = Number($selected.attr('data-offset'));

#### var commong\_media\_args = {

#### style : $selected.attr('style'),

#### filepath : $selected.attr('data-filepath'),

#### filters : $selected.attr('data-filters'),

#### hasaudio : $selected.attr('data-hasaudio'),

#### hasvideo : $selected.attr('data-hasvideo'),

#### type : $selected.attr('data-type'),

#### };

#### if ( action == 'remove-selected' ) {

#### if ( selection\_data.start > 0 ) {

#### var args = commong\_media\_args;

#### args.start = media\_start;

#### args.offset = media\_offset;

#### args.duration = selection\_data.start;

#### media\_arr.push( $.editor.createLayerMedia(args) );

#### }

#### if ( selection\_data.end < media\_duration ) {

#### var args = commong\_media\_args;

#### args.start = media\_start + selection\_data.start;

#### args.duration = media\_duration - selection\_data.end;

#### args.offset = media\_offset + selection\_data.end;

#### media\_arr.push( $.editor.createLayerMedia(args) );

#### }

#### }

#### else if ( action == 'crop' ) {

#### var args = commong\_media\_args;

#### args.start = media\_start;

#### args.duration = selection\_data.end - selection\_data.start;

#### args.offset = media\_offset + selection\_data.start;

#### media\_arr.push( $.editor.createLayerMedia(args) );

#### }

#### else if ( action == 'copy' ) {

#### var args = commong\_media\_args;

#### args.start = media\_start;

#### args.duration = selection\_data.end - selection\_data.start;

#### args.offset = media\_offset + selection\_data.start;

#### media\_arr.push( $.editor.createLayerMedia(args) );

#### }

#### var $layer = $selected.closest('.layer');

#### if ( media\_arr.length ) {

#### for ( var i = 0; i < media\_arr.length; i++ ) {

#### $layer.append(media\_arr[i]);

#### $.editor.initLayerMedia(media\_arr[i]);

#### }

#### media\_arr[0].trigger('click');

#### if ( action != 'copy' ) {

#### $selected.remove();

#### }

#### }

#### $('#preview-current-area .timeline-selection').remove();

#### $.editor.updateState();

#### },

#### applyFilters: function () {

#### var $selected\_media = $('.layer-media.selected');

#### if ( $selected\_media.length ) {

#### var preview\_video = document.getElementById('preview-current');

#### var filter;

#### var value;

#### var media\_type = $selected\_media.attr('data-type');

#### if ( media\_type != 'audio' ) {

#### $.editor.setFilter( $selected\_media, 'text', {

#### text : $('#media-text').val(),

#### fontsize : $('#media-text-size').val() || 24,

#### fontcolor\_expr : $('#media-text-color').val(),

#### y : $('#media-text-top').val(),

#### x : $('#media-text-left').val(),

#### box : $('#media-text-box').is(':checked'),

#### boxcolor : $('#media-text-boxcolor').val()

#### });

#### var $eq\_filters = $('.eq-filter');

#### for ( var i = 0; i < $eq\_filters.length; i++ ) {

#### $eq\_filter = $($eq\_filters[i]);

#### filter = $eq\_filter.attr('id');

#### value = $eq\_filter.val();

#### $.editor.setFilter( $selected\_media, filter, value );

#### }

#### }

#### if ( media\_type == 'audio' || media\_type == 'video' ) {

#### 

#### reverse\_value = $('#reverse-checkbox').is(':checked');

#### $.editor.setFilter( $selected\_media, 'reverse', reverse\_value );

#### $.editor.setFilter( $selected\_media, 'volume', $('#volume').val() );

#### }

#### if ( media\_type == 'blank' ) {

#### var $handle = $('#preview-current').closest('.preview-video-handle');

#### var color = $('#media-background-color').val();

#### $handle.css('background-color', color );

#### $.editor.setFilter( $selected\_media, 'background\_color', color.replace('#', '') );

#### }

#### if ( media\_type == 'audio' ) {

#### }

#### 

#### $.editor.preview\_timeline.setTimelineMode( true, preview\_video );

#### $.editor.updateState( $selected\_media );

#### }

#### },

#### setFilter : function( $media, filter, value ) {

#### var filters = $media.attr('data-filters');

#### if ( filters ) {

#### filters = JSON.parse(filters);

#### filters[filter] = value;

#### }

#### else {

#### filters = {};

#### filters[filter] = value;

#### }

#### 

#### filters = JSON.stringify(filters);

#### $media.attr('data-filters', filters);

#### },

#### getFilter : function( $media, filter ) {

#### var filters = $media.attr('data-filters');

#### if ( filters ) {

#### filters = JSON.parse(filters);

#### if ( filter ) {

#### return filters[filter];

#### }

#### else {

#### return filters;

#### }

#### }

#### return {};

#### },

#### applyOverlayPosition : function ( $media ) {

#### if ( $media ) {

#### var $handle = $('#preview-current').closest('.preview-video-handle');

#### var $parent = $handle.parent();

#### 

#### var total\_width = $parent.width();

#### var total\_height = $parent.height();

#### var width = $handle.width();

#### var height = $handle.height();

#### var left = $handle.offset().left - $parent.offset().left;

#### var top = $handle.offset().top - $parent.offset().top;

#### var overlay\_filters = $.editor.getFilter( $media, 'overlay' );

#### 

#### if ( !overlay\_filters ) {

#### overlay\_filters = {};

#### }

#### overlay\_filters.width = width \* 100 / total\_width;

#### overlay\_filters.height = height \* 100 / total\_height;

#### overlay\_filters.left = left \* 100 / total\_width;

#### overlay\_filters.top = top \* 100 / total\_height;

#### $.editor.setFilter( $media, 'overlay', overlay\_filters );

#### $.editor.updateState ( $media );

#### $.editor.preview\_timeline.setTimelineMode( true, document.getElementById('preview-current') );

#### $handle.css({

#### 'width': '100%',

#### 'height': '100%',

#### 'left': 0,

#### 'top': 0

#### });

#### $.editor.updateState( $media );

#### }

#### },

#### resetOverlayPosition : function ( $media ) {

#### var $handle = $('#preview-current').closest('.preview-video-handle'),

#### width = '100%',

#### height = '100%',

#### left = 0,

#### top = 0;

#### if ( $media ) {

#### var overlay\_filters = $.editor.getFilter( $media, 'overlay' );

#### 

#### overlay\_filters.width = width;

#### overlay\_filters.height = height;

#### overlay\_filters.left = left;

#### overlay\_filters.top = top;

#### $.editor.setFilter( $media, 'overlay', overlay\_filters );

#### $.editor.updateState ( $media );

#### }

#### 

#### $handle.css({

#### 'width': width,

#### 'height': height,

#### 'left': left,

#### 'top': top

#### });

#### },

#### resizePreviewScreens : function() {

#### $('.preview-video-holder').each( function(i, e) {

#### $e = $(e);

#### var width = $e.width();

#### $e.height(width \* $.editor.aspect\_ratio.height / $.editor.aspect\_ratio.width);

#### });

#### },

#### updateInterfaceElements : function() {

#### $selected\_media = $('.selected');

#### var media\_type = $selected\_media.attr('data-type');

#### $('#delete-media').attr('disabled', true);

#### var $preview\_area = $('#preview-current-area, .filters');

#### var $timeline = $preview\_area.find('.timeline');

#### $('.slider').slider('disable');

#### if ( $selected\_media.length ) {

#### $timeline.attr('data-disabled', false);

#### $timeline.attr('data-duration', $selected\_media.attr('data-duration'));

#### $timeline.attr('data-endtime', $selected\_media.attr('data-duration'));

#### $timeline.attr('data-offset', $selected\_media.attr('data-offset'));

#### // Media element from the library is selected

#### if ( $selected\_media.hasClass('media') ) {

#### $('#delete-media').attr('disabled', false);

#### $('#preview-current').attr('src', $selected\_media.attr('data-filepath'));

#### $preview\_area.find('button, input, textarea').attr('disabled', true);

#### $preview\_area.find('.preview-control-button').attr('disabled', false);

#### $timeline.attr('data-offset', '0');

#### $timeline.attr('data-zoom', '1');

#### }

#### // Media element from the timeline is selected

#### if ( $selected\_media.hasClass('layer-media') ) {

#### $('#preview-current-selection, #preview-current-remove-selections, #remove-selected, #remove-unselected, #copy-media').removeAttr('disabled');

#### if ( media\_type == 'audio' ) {

#### var filters\_holders = $('.audio-filters');

#### }

#### else if ( media\_type == 'blank' ) {

#### var filters\_holders = $('.blank-filters');

#### }

#### else if ( media\_type == 'video' ) {

#### var filters\_holders = $('.video-filters, .audio-filters');

#### $('#apply-overlay-position, #reset-overlay-position').removeAttr('disabled');

#### }

#### var filters = $.editor.getFilter( $selected\_media );

#### for ( var filter in filters ) {

#### if ( filters.hasOwnProperty(filter) ) {

#### if (filter == 'text') {

#### $('#media-text').val(filters[filter].text);

#### $('#media-text-color').val(filters[filter].fontcolor\_expr);

#### $('#media-text-box').prop('checked', filters[filter].box);

#### $('#media-text-boxcolor').val(filters[filter].boxcolor);

#### $('#media-text-size').val(filters[filter].fontsize).siblings('.slider').slider( 'option', 'value', filters[filter].fontsize );

#### $('#media-text-left').val(filters[filter].x);

#### $('#media-text-top').val(filters[filter].y);

#### }

#### else {

#### var input = $('#' + filter).val( filters[filter] );

#### var slider = input.siblings('.slider');

#### if( slider.length ) {

#### slider.slider( 'option', 'value', filters[filter] );

#### }

#### }

#### }

#### }

#### $preview\_area.find('.preview-control-button').attr('disabled', false);

#### $('#apply-fitlers').attr('disabled', false);

#### filters\_holders.find('.slider').slider('enable');

#### filters\_holders.find('button, input, textarea').attr('disabled', false);

#### $('#preview-current').attr('src', $selected\_media.attr('data-filepath'));

#### $('#preview-final').attr('src', '');

#### var is\_blank = ( media\_type == 'blank' );

#### if ( $selected\_media.attr('data-filters') || is\_blank ) {

#### $.editor.preview\_timeline.setTimelineMode ( true, document.getElementById('preview-current') );

#### }

#### else {

#### $.editor.preview\_timeline.setTimelineMode ( false, document.getElementById('preview-current') );

#### document.getElementById('preview-current').currentTime = $selected\_media.attr('data-offset') || 0;

#### }

#### }

#### }

#### // There is nothing selected

#### else {

#### $preview\_area.find('button, input, textarea').attr('disabled', 'true');

#### $('#preview-current').attr('src', '');

#### $preview\_area.find('button').attr('disabled', 'true');

#### $timeline.attr('data-disabled', 'true');

#### $timeline.attr('data-duration', '12');

#### $timeline.attr('data-endtime', '12');

#### $timeline.attr('data-offset', '0');

#### $timeline.attr('data-zoom', '1');

#### }

#### 

#### $.editor.preview\_timeline.reinit();

#### }

#### }

#### $.timeline = function(timeline, args) {

#### this.renderRuler = function() {

#### var duration\_step = this.duration / this.ruler\_markers;

#### var $ruler = $('<div class="timeline-ruler"></div>');

#### var $marker, time, time\_string;

#### for (var i = 1; i < this.ruler\_markers; i++) {

#### time = i \* duration\_step;

#### time = time + ""; // Convert to a string

#### $marker = $('<div class="ruler-marker"></div>')

#### $marker.css('left', (time \* 100 / this.duration) + "%");

#### if (i % 5 == 0) {

#### $marker.addClass('big');

#### }

#### if (i % ( 5 \* this.marker\_density ) == 0) {

#### $marker.text(time.toHHMMSS());

#### }

#### $ruler.append($marker);

#### }

#### return $ruler;

#### };

#### this.addLayer = function(){

#### var $layer = $('<div class="layer"></div>');

#### this.timeline.append( $layer );

#### this.renderLayers( $layer );

#### return $layer;

#### },

#### this.render = function() {

#### this.timeline.find('.time-marker').remove();

#### var time\_marker = this.timeline.find('.time-marker');

#### if (!time\_marker.length) {

#### time\_marker = $('<div class="time-marker"><div class="line"></div></div>');

#### this.timeline.prepend(time\_marker);

#### if ( this.timeline.attr('data-disabled') != "true" ) {

#### var args = {

#### axis: "x",

#### containment: "parent",

#### scroll: true

#### };

#### 

#### if (this.options && this.options.time\_marker\_drag) {

#### args.drag = this.options.time\_marker\_drag;

#### }

#### 

#### if (this.options && this.options.time\_marker\_start) {

#### args.start = this.options.time\_marker\_start;

#### }

#### if (this.options && this.options.time\_marker\_stop) {

#### args.stop = this.options.time\_marker\_stop;

#### }

#### time\_marker.draggable(args);

#### }

#### else {

#### time\_marker.css('left', '0');

#### }

#### }

#### this.renderLayers();

#### },

#### this.renderLayers = function( layers, droppable ) {

#### if ( !layers ) {

#### layers = this.timeline.find('.layer');

#### }

#### if ( this.options && this.options.droppable ) {

#### layers.droppable({

#### accept: ".layer-media, .media",

#### drop: function( event, ui ) {

#### $this = $(this);

#### var left\_px = ui.position.left;

#### var e;

#### if ( ui.helper && ui.helper !== ui.draggable ) {

#### e = ui.helper.clone();

#### $.editor.initLayerMedia(e);

#### var offset = $this.offset();

#### left\_px -= offset.left;

#### }

#### else {

#### e = ui.draggable;

#### }

#### var left\_percent = (left\_px \* 100) / Number($this.width());

#### var new\_start = (left\_percent / 100) \* Number($.editor.timeline.duration);

#### e.css('left', left\_percent + "%");

#### e.attr('data-start', new\_start);

#### e.css('top', 0).detach().appendTo($this);

#### $.editor.updateState();

#### }

#### });

#### }

#### 

#### var tl = this;

#### layers.each(function(i,e) {

#### $e = $(e);

#### // set width

#### $e.css('width', tl.layer\_width + "%");

#### // render ruler

#### $e.find('.timeline-ruler').remove();

#### $e.append ( tl.renderRuler() );

#### });

#### // Process the media elements

#### $.editor.initLayerMedia();

#### };

#### this.reinit = function() {

#### this.zoom = timeline.attr('data-zoom') || 1;

#### this.marker\_density = timeline.attr('data-markerdensity') || 60;

#### this.duration = timeline.attr('data-duration') || 600;

#### this.layer\_width = timeline.attr('data-layerwidth') || 100;

#### this.ruler\_markers = parseInt(Math.round(this.zoom) \* this.layer\_width \* this.marker\_density);

#### this.initial\_layer\_width = this.layer\_width;

#### this.render();

#### };

#### this.animateTimeMarker = function() {

#### var $timeline = $(this.timeline);

#### var $time\_marker = $timeline.find('.time-marker');

#### var duration = Number( $timeline.attr('data-duration') );

#### var end\_time = Number( $timeline.attr('data-endtime') );

#### var position\_left = Number($time\_marker.css('left').replace('px', ''));

#### var layer\_width = Number($timeline.find('.layer').width());

#### var current\_time = position\_left \* duration / layer\_width;

#### var end\_position = layer\_width \* end\_time / duration;

#### var animation\_time = end\_time - current\_time;

#### $time\_marker.animate({

#### left: end\_position + "px"

#### }, animation\_time \* 1000, 'linear');

#### };

#### this.stopTimeMarker = function( current\_time ) {

#### var $timeline = $(this.timeline);

#### var $time\_marker = $timeline.find('.time-marker');

#### var offset = Number( $timeline.attr('data-offset') ) || 0;

#### current\_time -= offset;

#### if ( current\_time < 0 ) {

#### current\_time = 0;

#### }

#### var end\_time = Number($timeline.attr('data-endtime')) || 0;

#### var duration = Number($timeline.attr('data-duration')) || 1;

#### var end\_position = Number($timeline.find('.layer').width()) \* end\_time / duration;

#### var current\_time\_percent = current\_time \* 100 / end\_time;

#### $time\_marker.stop().css('left', (current\_time\_percent \* end\_position / 100) + "px");

#### };

#### this.getSelectionData = function() {

#### var $timeline = $(this.timeline);

#### var $selections = $timeline.find('.timeline-selection');

#### var selections\_data = [];

#### var duration = $timeline.attr('data-duration');

#### var layer\_width = Number($timeline.find('.layer').width());

#### 

#### for (var i = 0; i < $selections.length; i++ ) {

#### $selection = $($selections[i]);

#### var left\_edge = $selection.position().left;

#### var right\_edge = left\_edge + $selection.width();

#### selections\_data.push({

#### start: left\_edge \* duration / layer\_width,

#### end: right\_edge \* duration / layer\_width,

#### });

#### }

#### return selections\_data;

#### }

#### this.setTimelineMode = function ( stream, video ) {

#### video.stream = stream;

#### video.timelineTime = 0;

#### video.currentTime = 0;

#### video.prevTime = 0;

#### video.pause();

#### this.stopTimeMarker( 0 );

#### if ( stream ) {

#### video.src = "/preview";

#### }

#### }

#### this.timeline = timeline;

#### if (!args) {

#### args = {};

#### }

#### this.options = {

#### droppable : args.droppable || false,

#### time\_marker\_drag: args.time\_marker\_drag,

#### time\_marker\_start: args.time\_marker\_start,

#### time\_marker\_stop: args.time\_marker\_stop

#### }

#### this.reinit();

#### if (args.zoom\_button) {

#### var tl = this;

#### args.zoom\_button.on('click', function(){

#### $this = $(this);

#### var amount = Number($this.attr('data-amount'));

#### var current\_zoom = Number(tl.zoom);

#### var new\_zoom = Math.round((current\_zoom + amount) \* 100) / 100 ; // Remove inaccuracies from using float numbers

#### 

#### if ( new\_zoom > 0.1 && new\_zoom < 10 ) {

#### tl.zoom = new\_zoom;

#### tl.layer\_width = tl.initial\_layer\_width \* new\_zoom;

#### tl.render();

#### }

#### });

#### }

#### return this;

#### }

#### }( jQuery ));

#### $(function(){

#### $.editor.init();

#### });

#### // Converts a string representing a number of seconds to the format hh:mm:ss

#### String.prototype.toHHMMSS = function () {

#### var sec\_num = parseInt(this, 10);

#### var hours = Math.floor(sec\_num / 3600);

#### var minutes = Math.floor((sec\_num - (hours \* 3600)) / 60);

#### var seconds = sec\_num - (hours \* 3600) - (minutes \* 60);

#### if (hours < 10) {hours = "0"+hours;}

#### if (minutes < 10) {minutes = "0"+minutes;}

#### if (seconds < 10) {seconds = "0"+seconds;}

#### return hours+':'+minutes+':'+seconds;

#### }