

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



- Πτυχιακή Εργασία -

Δημιουργία ψυχαγωγικής web εφαρμογής καταγραφής
και τροποποίησης βίντεο σε real time

Ιωάννης Νικολούλης
Επιβλέπων: Μιχαήλ Στεφανιδάκης

Μάιος 2024

Επιβλέπων

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής,
Τμήμα Πληροφορικής, Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής,
Τμήμα Πληροφορικής, Ιόνιο Πανεπιστήμιο

Ελένη Χριστοπούλου, ΕΔΙΠ (Εργαστηριακό Διδακτικό Προσωπικό),
Τμήμα Πληροφορικής, Ιόνιο Πανεπιστήμιο

Θεόδωρος Ανδρόνικος, Καθηγητής,
Τμήμα Πληροφορικής, Ιόνιο Πανεπιστήμιο

Περίληψη

Στη σημερινή εποχή παρατηρείται η ολοένα αυξανόμενη χρήση των μέσων κοινωνικής δικτύωσης, κάτι που επιφέρει και την είσοδο νέων τεχνολογιών που αποσκοπούν στη συνεχή ψυχαγωγία των χρηστών. Ανέκαθεν, οι πλατφόρμες αυτές είχαν ως σκοπό τη σύνδεση των χρηστών μεταξύ τους δίνοντας τους τα απαραίτητα εργαλεία ώστε να εξατομικεύουν την παρουσία τους στον διαδικτυακό χώρο σύμφωνα με τις δικές τους ανάγκες και προτιμήσεις. Πλέον, σε κάθε τέτοια πλατφόρμα οι χρήστες έχουν πρόσβαση σε χιλιάδες εικονικά φίλτρα τα οποία μπορούν να εφαρμόσουν πάνω στο πρόσωπό τους και να δημιουργήσουν εικόνες και βίντεο που μπορούν να συμπεριλάβουν σε μηνύματα και αναρτήσεις. Στην παρούσα εργασία θα αναφερθούμε στην διαδικασία που χρειάζεται να ακολουθήσουμε για την ανάπτυξη μιας web εφαρμογής καταγραφής και τροποποίησης βίντεο σε πραγματικό χρόνο. Συγκεκριμένα, θα εξετάσουμε τις τεχνολογίες που αξιοποιεί η βιβλιοθήκη Clmtrackr για τον εντοπισμό και την τροποποίηση των χαρακτηριστικών προσώπων, καθώς και τον κώδικα που χρησιμοποιείται για την υλοποίηση της πάνω σε μια δυναμική ιστοσελίδα. Παράλληλα, θα ασχοληθούμε με τη διαμόρφωση των σελίδων της εφαρμογής προσθέτοντας λειτουργίες όπως η εγγραφή και σύνδεση χρηστών και ρυθμίζοντας την εμφάνισή της με σκοπό τη δημιουργία μιας ευχάριστης εμπειρίας κατά τη χρήση της. Γενικότερα, σκοπός αυτής της προσπάθειας είναι να δημιουργηθεί μια εξοικείωση με τον συνεχώς εξελισσόμενο κόσμο της τροποποίησης βίντεο σε πραγματικό χρόνο, αξιοποιώντας μεθόδους και τεχνικές απλές ως προς την υλοποίηση και τροποποίησή τους οι οποίες απαιτούν ελάχιστους πόρους και υπολογιστική δύναμη.

Abstract

Nowadays social media platforms seem to be gaining increasing popularity, which in return introduces new technologies which aim to enhance the entertainment when being navigated by the users. Since their creation, these platforms have aimed to allow the users to connect with one another while providing them with the necessary tools in order to create their own personal changes to the user experience according to their own tastes and needs. At the moment, users have access to thousands of face filters which can be applied to their faces and create images and videos which can be sent via private messages or posted publicly online. In this project we will describe the procedure which takes place in the development of a real time video capture and editing web application. Specifically, we will take a look into the technologies used by the Clmtrackr library for face tracking and the modification of facial features, as well as the code being used in order to embed it into a dynamic webpage. Meanwhile, we will embark on the formatting of the app's pages by adding features such as user register and login and by fine tuning the application's appearance and user interface with the goal to create a pleasant user experience. In general, the aim of this project is to acquire familiarity with the rapidly advancing world of real time video editing while making use of easy to implement modifiable methods and techniques requiring minimum resources and computer power.

Περιεχόμενα

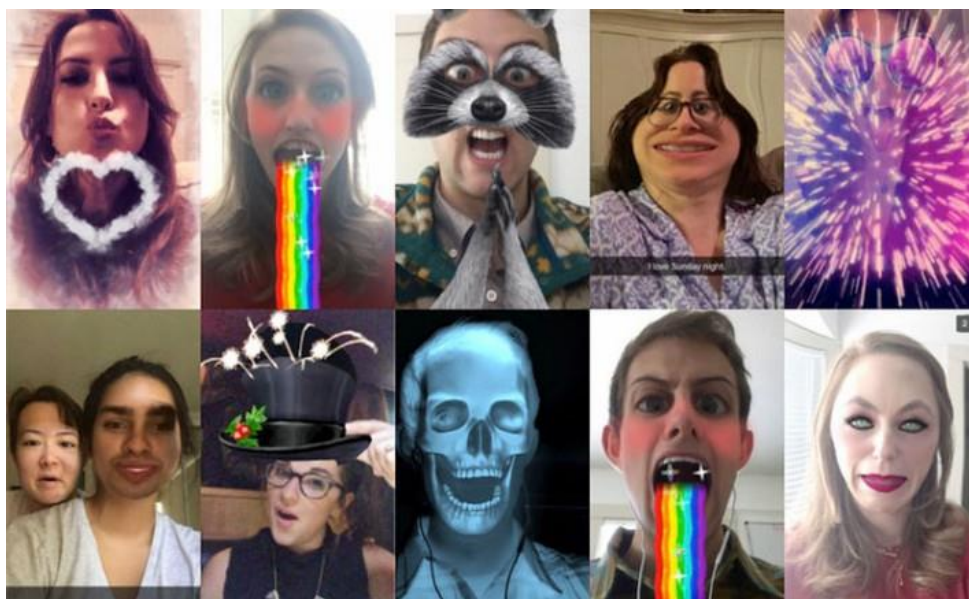
1	Εισαγωγή	1
2	Αλγόριθμοι, APIs και Βιβλιοθήκες	4
2.1	Media Streams API	4
2.2	Canvas API	5
2.3	WebGL API	5
2.4	File API	6
2.5	Fetch API	6
2.6	JavaScript Module APIs	6
2.7	Βιβλιοθήκη Poisson Blending	7
2.8	Βιβλιοθήκη Clmtrackr	7
2.9	Βιβλιοθήκη face_deformer	12
3	Αρχιτεκτονική και Υλοποίηση της εφαρμογής	13
3.1	Διαδικασία του face substitution	13
3.2	Επιπλέον λειτουργίες της εφαρμογής	17
3.3	Frontend User Interface	22
3.4	Upload εικόνας	25
3.5	Επεξεργασία της ανεβασμένης εικόνας	26
3.6	Client.js script	30
3.7	Δεδομένα του getSelectData route	30
3.8	Δεδομένα του getImages route	32
3.9	Container με τις διαθέσιμες εικόνες	35
3.10	Deployment/Εγκατάσταση	39
4	Συμπεράσματα και Μελλοντικές Επεκτάσεις	40
5	Βιβλιογραφία/Δικτυογραφία	41

1 Εισαγωγή

Η ραγδαία ανάπτυξη και διάδοση των μέσων κοινωνικής δικτύωσης έχει οδηγήσει στην αναζήτηση νέων εργαλείων που αποσκοπούν στη διατήρηση του ενδιαφέροντος των χρηστών κατά τη χρήση των εφαρμογών τους. Βασικό κομμάτι στην αύξηση του ψυχαγωγικού χαρακτήρα των εφαρμογών αυτών αποτελούν τα εφέ και τα φίλτρα στα οποία έχουν πρόσβαση οι χρήστες σε πραγματικό χρόνο μέσω της κάμερας της συσκευής τους. Περισσότερο κοινά και ελαφριά κατά την εκτέλεσή τους είναι τα κλασικά φίλτρα, τα οποία τροποποιούν τη συνολική εικόνα που επιστρέφεται από την κάμερα με την προσθήκη φίλτρων, όπως το blur ή το vignette, καθώς και με την αυξομείωση τιμών όπως του brightness και του saturation. Μέχρι και το 2015 τα φίλτρα αυτά αποτελούσαν το μοναδικό εργαλείο τροποποίησης βίντεο και εικόνων στα social media. Σήμερα, με την ευρεία χρήση τεχνολογιών όπως η Επαυξημένη Πραγματικότητα και η Μηχανική Μάθηση οι χρήστες έχουν πρόσβαση σε χιλιάδες δισδιάστατα και τρισδιάστατα face filters (φίλτρα προσώπου) τα οποία ανιχνεύουν τα χαρακτηριστικά του προσώπου και εφαρμόζονται πάνω σε αυτό τροποποιώντας τις τιμές τους (όπως το μέγεθος και την περιστροφή τους) σύμφωνα με τις κινήσεις του χρήστη σε πραγματικό χρόνο.



Εικόνα 1.1 Λίστα με τα αρχικά φίλτρα του Instagram το 2011 [1].



Εικόνα 1.2 Λίστα με face filters του Snapchat από το 2015 και έπειτα [2].

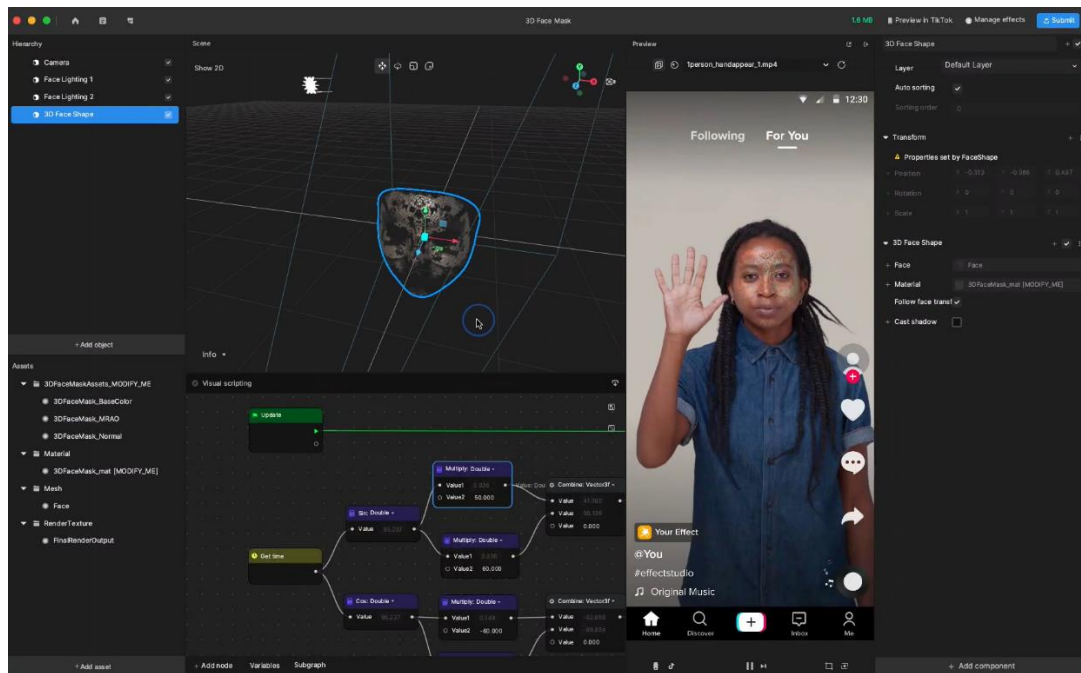
Έτσι, στα δημοφιλή μέσα κοινωνικής δικτύωσης (Instagram, TikTok, Snapchat) παρατηρείται ότι ακολουθείται παρόμοια μέθοδος για τη δημιουργία και διαχείριση των εφέ και των φίλτρων. Η εταιρεία κάθε εφαρμογής παρέχει στους χρήστες εξατομικευμένες πλατφόρμες με εύκολα στην κατανόηση γραφικά και μεθόδους υλοποίησης, ενώ παράλληλα προσφέρουν λεπτομερή εγχειρίδια και βίντεο tutorials. Η ευκολία στη χρήση των εφαρμογών αυτών τις καθιστούν κατάλληλες τόσο για επαγγελματική όσο και για καθημερινή χρήση συμβάλλοντας έτσι στην άκρως αυξημένη ανάπτυξη εφέ και φίλτρων από τους χρήστες.

Τα προγράμματα αυτά βασίζονται στις λειτουργίες της Επαυξημένης Πραγματικότητας (AR) χωρίς όμως οι εταιρείες να παρέχουν στο κοινό τους αλγορίθμους και τις μεθόδους που χρησιμοποιούν για τον συνεχή εντοπισμό του προσώπου σε κάθε frame (tracking) και την τροποποίηση της θέσης του φίλτρου στον χώρο ανάλογα με το πρόσωπο του χρήστη. Παράλληλα, διαθέτουν γλώσσες scripting τόσο οπτικές (Effect House¹ του TikTok) όσο και κειμένου (JavaScript στο Spark AR² του Facebook και στο Lens Studio³ του Snapchat) οι οποίες ξεκλειδώνουν τις πλήρεις δυνατότητες των εφαρμογών προσφέροντας στους χρήστες επιπλέον ευελιξία στην ανάπτυξη των εφέ και εμπλουτίζοντας τη διάδραση με το χρήστη. Βέβαια, αξίζει να σημειωθεί πως δεν είναι απαραίτητη η χρήση κώδικα για τη δημιουργία των εφέ.

¹ <https://effecthouse.tiktok.com>

² <https://spark.meta.com>

³ <https://ar.snap.com/lens-studio>



Εικόνα 1.3 Το περιβάλλον της εφαρμογής Effect House του TikTok [3].

Σκοπός, λοιπόν, της παρούσας εργασίας είναι η μελέτη των εργαλείων και των τεχνικών που απαιτούνται για την υλοποίηση μιας διαδικτυακής εφαρμογής καταγραφής και τροποποίησης βίντεο. Συγκεκριμένα, το περιβάλλον που θα δημιουργήσουμε θα παρέχει στον χρήστη, μέσω της τεχνικής του face tracking, τη δυνατότητα εφαρμογής διαφορετικών προσώπων πάνω στο δικό του σε πραγματικό χρόνο αξιοποιώντας τις δυνατότητες της βιβλιοθήκης Clmtrackr μέσω της οποίας θα εκτελεστεί το μεγαλύτερο μέρος των λειτουργιών της web εφαρμογής. Φυσικά, το αποτέλεσμα θα είναι αρκετά περιορισμένο με λιγότερες λειτουργίες και ακρίβεια από τα περιβάλλοντα εφαρμογής εφέ των δημοφιλών μέσων κοινωνικής δικτύωσης. Παρ' όλ' αυτά, στόχος είναι η φιλοσοφία της εφαρμογής να προσομοιώνει κατά το μέγιστο το ψυχαγωγικό χαρακτήρα και τη μοναδική εμπειρία που προσφέρουν στον χρήστη τα εργαλεία αυτά.

Στα κεφάλαια που ακολουθούν θα εξετάσουμε αρχικά τους αλγορίθμους, τις βιβλιοθήκες και τα APIs που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Στη συνέχεια, θα αναφερθούμε αναλυτικά στην αρχιτεκτονική και θα μελετήσουμε τον κώδικα πίσω από τις βασικές λειτουργίες της εφαρμογής, ενώ παράλληλα θα περιγράψουμε τη διαδικασία που απαιτείται για την εγκατάστασή της. Τέλος, θα γίνει αναφορά στα συμπεράσματα, τις μελλοντικές επεκτάσεις και όσα αποκομίστηκαν μέσα από τη μελέτη του συγκεκριμένου θέματος.

2 Αλγόριθμοι, APIs και Βιβλιοθήκες

Τα APIs ή διεπαφές προγραμματισμού εφαρμογών αποτελούν αναπόσπαστο κομμάτι του λογισμικού των υπολογιστών, καθώς παρέχουν στον προγραμματιστή τις υπηρεσίες που απαιτούνται για τη δημιουργία των λειτουργιών που επιθυμεί να συμπεριλάβει. Για την υλοποίηση των λειτουργιών των εφαρμογών ιστού χρησιμοποιούνται Web APIs που συνήθως καλούνται μέσα σε JavaScript. Στην προκειμένη περίπτωση, APIs περιλαμβάνονται τόσο σε HTML όσο και σε JavaScript εντός του αρχείου index.html (δηλαδή της αρχικής σελίδας) το οποίο κάνει host ο server.

2.1 Media Streams API

Η κύρια χρήση της εφαρμογής βασίζεται στο Media Capture and Streams API το οποίο παρέχει τις διεπαφές που σχετίζονται με την πρόσβαση στις οπτικοακουστικές λειτουργίες του υπολογιστή μέσω των συσκευών της κάμερας και του μικροφώνου. Συγκεκριμένα, για τη χρήση της κάμερας αξιοποιείται η διεπαφή MediaDevices μέσω της οποίας καλείται το `getUserMedia()`. Η συνάρτηση αυτή ζητά από τον χρήστη πρόσβαση στην κάμερα και το μικρόφωνο (εάν υπάρχουν) και παράγει ένα αντικείμενο `MediaStream` στο οποίο περιλαμβάνονται τα streams των πολυμέσων (εικόνα, ήχος) για τα οποία έχει επιτραπεί η πρόσβαση. Σε αυτήν την περίπτωση, καθώς μας ενδιαφέρει μόνο το stream της κάμερας προστίθεται η επιλογή `{ video: true }` η οποία περιορίζει το αίτημα πρόσβασης μόνο στη webcam του χρήστη. Εάν η κάμερα είναι διαθέσιμη και δοθεί πρόσβαση, καλείται μια callback συνάρτηση η οποία δέχεται ως παράμετρο το stream της κάμερας και το τοποθετεί στο video element που έχει δημιουργηθεί στην HTML. Ως αποτέλεσμα, ο χρήστης βλέπει σε πραγματικό χρόνο το stream της κάμερας στο πρόγραμμα περιήγησης της επιλογής του. Αξίζει να σημειωθεί, ότι καθώς οι παραπάνω λειτουργίες απαιτούν πρόσβαση σε προσωπικά δεδομένα του χρήστη η λειτουργία τους είναι διαθέσιμη μόνο σε ασφαλή περιβάλλοντα (εφαρμογές που γίνονται host σε secure server ή τοπικά σε localhost) κάτι που αποσκοπεί στην προστασία της ταυτότητας και των αρχείων του χρήστη από κινδύνους του διαδικτύου.

2.2 Canvas API

Βασικό κομμάτι της εφαρμογής αποτελούν τα πλαίσια που περιέχει η ιστοσελίδα για την εμφάνιση των οπτικών αποτελεσμάτων στο χρήστη. Το Canvas API αποτελεί σημαντικό εργαλείο για τη σχεδίαση 2D γραφικών σε HTML και JavaScript και καθίσταται απαραίτητο σε web εφαρμογές επεξεργασίας βίντεο σε πραγματικό χρόνο. Έτσι, εντός του index.html δημιουργούνται τρία canvas elements τα οποία στη συνέχεια τροποποιούνται και εμπλουτίζονται μέσα σε scripts. Στο videocanvas τοποθετείται το stream της κάμερας που λαμβάνεται από το getUserMedia() πριν από οποιαδήποτε επεξεργασία. Το maskcanvas περιέχει τη μάσκα στην οποία διαχωρίζεται (με άσπρο χρώμα) το πρόσωπο από το φόντο της εικόνας που θέλουμε να εφαρμόσουμε και τέλος, το newcanvas αποτελεί το τελικό αποτέλεσμα που εμφανίζεται στο χρήστη με το παραποιημένο, με βάση τη θέση των χαρακτηριστικών του προσώπου του χρήστη, πρόσωπο της εικόνας επικολλημένο στο videocanvas.

2.3 WebGL API

Η διαδικασία της παραμόρφωσης και της εφαρμογής του νέου προσώπου με βάση τις συντεταγμένες των σημείων των χαρακτηριστικών του προσώπου του χρήστη εκτελείται με τη βοήθεια της βιβλιοθήκης face_deformer.js. Η βιβλιοθήκη αυτή υλοποιείται πάνω σε canvas elements που αξιοποιούν τις δυνατότητες του WebGL με αποτέλεσμα η υποστήριξη του WebGL API να είναι απαραίτητη τόσο από το πρόγραμμα περιήγησης όσο και από το hardware του υπολογιστή. Το WebGL επιτρέπει στην εφαρμογή να αξιοποιήσει τεχνικές hardware acceleration μέσω της GPU επιταχύνοντας δραστικά τη διαδικασία του real-time video processing. Έτσι, πριν από οποιαδήποτε λειτουργία ελέγχεται αν υποστηρίζεται η διεπαφή WebGLRenderingContext, η οποία στη συνέχεια δημιουργεί δύο πλαίσια σχεδίασης γραφικών πάνω στα canvas elements webgl_overlay και webgl_overlay2 και τα αναθέτει στις μεταβλητές WebGLContext και WebGLContext2 αντίστοιχα. Η χρήση των δύο τελευταίων μεταβλητών είναι ιδιαίτερα σημαντική, καθώς μέσω αυτών ορίζεται το παράθυρο (δηλαδή οι διαστάσεις του καμβά) στο οποίο θέλουμε να απεικονιστούν τα αποτελέσματα του WebGL επιταχύνοντας σημαντικά τη διαδικασία.

2.4 File API

Η δυνατότητα που δίνεται στον χρήστη να ανεβάζει δικές του φωτογραφίες προσώπων και να τις χρησιμοποιεί στο face substitution δημιουργεί την ανάγκη χρήσης του File API το οποίο προσφέρει τη δυνατότητα στη διαδικτυακή εφαρμογή να έχει πρόσβαση στα περιεχόμενα των φακέλων του χρήστη. Εντός του index.html δημιουργείται `<form>` το οποίο περιέχει ένα file `<input>` element το οποίο (όταν πατηθεί το κουμπί Upload) εμφανίζει στο χρήστη το παράθυρο με τα αρχεία του υπολογιστή. Προκειμένου, βέβαια, η εικόνα να ανέβει και να τοποθετηθεί τόσο σε φάκελο στον κατάλογο της εφαρμογής όσο και ως καταχώρηση στη βάση δεδομένων χρειάζεται να πραγματοποιηθεί επικοινωνία μεταξύ του index.html και του server. Όταν ο χρήστης επιλέξει την εικόνα που επιθυμεί στέλνεται ως request στον server με τη μέθοδο HTTP POST και στη συνέχεια ο server πραγματοποιεί τα απαραίτητα βήματα για την αποθήκευση του αρχείου. Αναλυτικότερος σχολιασμός σχετικά με τις λειτουργίες και τον κώδικα του server θα πραγματοποιηθεί σε επόμενο κεφάλαιο.

2.5 Fetch API

Εντός του index.html γίνεται κλήση στο εξωτερικό JavaScript αρχείο client.js το οποίο δρα ως διαμεσολαβητής μεταξύ του index.html και του server. Μέσω του client.js μεταφέρονται οι πληροφορίες των εικόνων που έχει ανεβάσει κάθε χρήστης από τη βάση δεδομένων με σκοπό την επεξεργασία και εμφάνισή τους στο πρόγραμμα περιήγησης του χρήστη. Για να επιτευχθεί η λειτουργία αυτή χρησιμοποιείται το Fetch API το οποίο αξιοποιώντας ασύγχρονες συναρτήσεις στέλνει POST requests στο server της εφαρμογής και λαμβάνει στοιχεία (όπως το όνομα και η τοποθεσία εντός του καταλόγου) σχετικά με τις προσωπικές εικόνες που έχει ανεβάσει ο εκάστοτε χρήστης με σκοπό τη χρήση και επεξεργασία τους από το index.html.

2.6 JavaScript Module APIs

Η εφαρμογή εκτελείται μέσω του αρχείου server.js το οποίο δημιουργεί τον τοπικό server μέσω του Node.js. Για να επιτευχθούν οι λειτουργίες του server είναι απαραίτητη η εγκατάσταση και χρήση modules που παρέχει το Node. Τα modules αυτά περιέχουν APIs, όπως το express.js API που προσφέρει τις middleware συναρτήσεις για τη δημιουργία της εφαρμογής, το SQLite3

API για τη διάδραση με SQLite βάσεις δεδομένων και το multer API για τη μεταφόρτωση αρχείων.

2.7 Βιβλιοθήκη Poisson Blending

Όπως θα δούμε στη συνέχεια, προκειμένου το αποτέλεσμα της αλλαγής του προσώπου να είναι ρεαλιστικό και φυσικό στο ανθρώπινο μάτι, χρησιμοποιείται η τεχνική του Poisson Blending. Ο δημιουργός του Clmtrackr έχει συμπεριλάβει μια ήδη υπάρχουσα βιβλιοθήκη⁴ γραμμένη σε JavaScript, η οποία χρησιμοποιώντας το Canvas API εφαρμόζει επαναληπτικά τον αλγόριθμο του Poisson επί του νέου προσώπου και του χρήστη, έως ότου να επιτευχθεί μία ομαλή ένωση των δύο εικόνων.

2.8 Βιβλιοθήκη Clmtrackr

Το Clmtrackr αποτελεί μια βιβλιοθήκη ανοιχτού κώδικα γραμμένη σε JavaScript που επιτυγχάνει την ακριβή εύρεση των θέσεων των χαρακτηριστικών (facial features) ενός προσώπου ή αλλιώς το ακριβές tracking προσώπων σε πραγματικό χρόνο. Η λειτουργία της επιτυγχάνεται μέσω της εκπαίδευσης μοντέλων από τη βάση δεδομένων προσώπων του MUCT⁵ σε συνδυασμό με προσωπικές εικόνες από το δημιουργό. Καθώς δεν παρέχεται περιβάλλον εργασίας η επιλογή αυτή είναι περισσότερο κατάλληλη για προγραμματιστές που θέλουν να τη συμπεριλάβουν στη διαδικτυακή εφαρμογή τους. Οι κύριες χρήσεις της περιλαμβάνουν την εφαρμογή διαφορετικού προσώπου μέσω εικόνας στο εντοπιζόμενο πρόσωπο (face substitution), την ανίχνευση των συναισθημάτων του εντοπιζόμενου προσώπου (emotion detection), καθώς και την παραμόρφωση του προσώπου μέσω της τροποποίησης των χαρακτηριστικών του (face deformation). Στην πρωτότυπη μορφή της, η βιβλιοθήκη δεν αποτελεί τόσο ακριβή και αποδοτική λύση για face tracking όσο τα προγράμματα των μέσων κοινωνικής δικτύωσης. Παρ'ολ'αυτά ο δημιουργός παρέχει στους χρήστες εργαλεία για την εκπαίδευση μοντέλων με σκοπό τη βελτίωση της ακρίβειας κατά τον εντοπισμό του προσώπου. Σε ιδανικές συνθήκες καθαρής εικόνας και όμοιου φωτισμού ο εντοπισμός προσώπου μέσω της βιβλιοθήκης είναι σχεδόν ισότιμος και αποτελεσματικός όσο και των σύνθετων και

⁴ https://github.com/wellflat/imageprocessing-labs/tree/main/cv/poisson_blending

⁵ <http://www.milbo.org/muct/>

εξελιγμένων αλγορίθμων Επαυξημένης Πραγματικότητας των social media. Όσον αφορά την εξατομίκευση, μαζί με τη βιβλιοθήκη και τα παραδείγματα χρήσης της παρέχονται και τα Clmtools τα οποία περιλαμβάνουν μεταξύ άλλων κώδικα και σελίδα σε browser (image annotator) για τον εντοπισμό των χαρακτηριστικών του προσώπου σε εικόνες τις οποίες μπορεί να ανεβάσει ο χρήστης. Η λειτουργία αυτή αξιοποιεί το Clmtrackr για την παραγωγή των facial features του προσώπου της εικόνας και δίνει τη δυνατότητα στον χρήστη, με την εξαγωγή των συντεταγμένων των σημείων των χαρακτηριστικών, να χρησιμοποιήσει τις δικές του εικόνες στο face substitution.

Το Clmtrackr χρησιμοποιεί τον αλγόριθμο CLM⁶ (Constrained Local Model) για την εύρεση των χαρακτηριστικών του προσώπου τόσο σε στατικές εικόνες όσο και σε εγγεγραμμένα και σε πραγματικό χρόνο βίντεο. Ο CLM δημιουργεί ένα appearance model το οποίο εκπαιδεύεται μέσω Μηχανικής Μάθησης πάνω σε εικόνες ανθρωπίνων προσώπων και χρησιμοποιώντας επαναληπτικές μεθόδους παράγει κατάλληλα templates για τα facial landmarks (μάτια, μύτη, στόμα κ.τ.λ.) κάθε εικόνας του σετ δεδομένων εκπαίδευσης. Έπειτα, με την εφαρμογή του εκπαιδευμένου μοντέλου επιτυγχάνεται ακριβής εντοπισμός των χαρακτηριστικών σε σχεδόν οποιοδήποτε πρόσωπο εικόνας ή βίντεο.

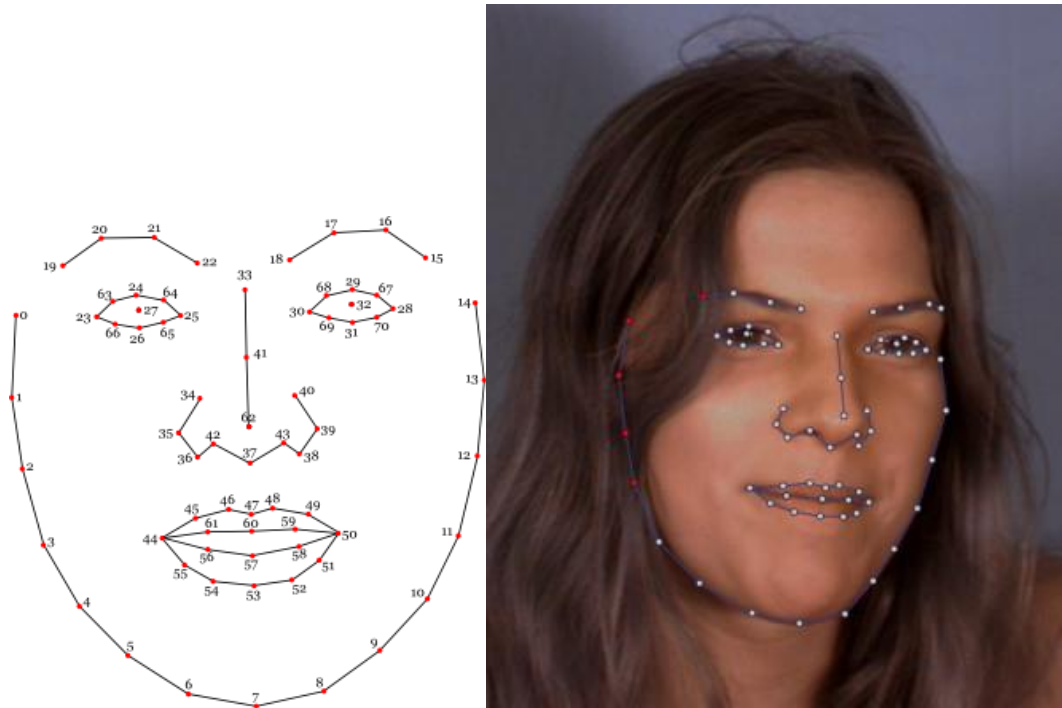
Κατά τη διαδικασία του face substitution απαιτούνται δύο βήματα. Αρχικά, χρησιμοποιώντας το CLM και το εργαλείο image annotator πραγματοποιείται εξαγωγή των σημείων των χαρακτηριστικών του προσώπου της εικόνας που θέλουμε να χρησιμοποιήσουμε. Έπειτα, αξιοποιώντας τις συντεταγμένες των σημείων αυτών γίνεται κατάλληλη παραμόρφωση του εικονιζόμενου προσώπου σύμφωνα με το σχήμα του εντοπιζόμενου προσώπου με σκοπό την επικάλυψή του στην ακριβή πόζα και θέση του. Η διαδικασία αυτή συμβαίνει μέσω του face_deformer.js το οποίο συμπεριλαμβάνεται στο GitHub της βιβλιοθήκης. Βέβαια, για να επιτευχθεί ένα πιο φυσικό αποτέλεσμα απαιτείται ένα επιπλέον βήμα το οποίο εξασφαλίζει την αντιστοίχιση των χρωμάτων της εικόνας με εκείνα του βίντεο το οποίο επικαλύπτει. Για το αποτέλεσμα αυτό χρησιμοποιείται η μέθοδος Poisson Blending⁷ η οποία ομαλοποιεί τις χρωματικές διαβαθμίσεις (color gradients) της επικολλημένης εικόνας με βάση το φόντο στο οποίο εφαρμόζεται. Η διαδικασία αυτή επαναλαμβάνεται και για τα color gradients της υπόλοιπης εικόνας με αποτέλεσμα το Poisson Blending να αποτελεί μια αρκετά χρονοβόρα,

⁶ <https://personalpages.manchester.ac.uk/staff/timothy.f.cootes/Models/clm.html>

⁷ https://erkaman.github.io/posts/poisson_blending.html

αλλά αναγκαία, διαδικασία, καθώς η ένωση των δύο προσώπων σε ένα ενιαίο επιφέρει επαρκώς ακριβή και ρεαλιστικά αποτελέσματα.

Κατά την έναρξή της η εφαρμογή εντοπίζει τις ακριβείς θέσεις των χαρακτηριστικών του προσώπου που έχει επιλέξει ο χρήστης είτε από εικόνα είτε από βίντεο. Για να επιτευχθεί η λειτουργία αυτή, χρησιμοποιείται η βιβλιοθήκη Clmtrackr.js ρόλος της οποίας είναι η εφαρμογή (fitting) ενός σταθερού μοντέλου σε κάθε πρόσωπο με σκοπό την εξαγωγή και τον εντοπισμό των χαρακτηριστικών του. Το μοντέλο αυτό αποτελείται από 70 σημεία (facial landmarks) καθένα από τα οποία αναπαριστά τα μέρη των χαρακτηριστικών του προσώπου στο οποίο εφαρμόζεται (για παράδειγμα απαιτούνται 4 σημεία για να αποτυπωθεί κάθε φρύδι ή 9 σημεία για κάθε μάτι). Το fitting του μοντέλου πραγματοποιείται με τον αλγόριθμο CLM ο οποίος ξεκινά με μία εκτίμηση της θέσης των χαρακτηριστικών του προσώπου της εικόνας ή του βίντεο με τη χρήση ενός προεκπαιδευμένου μοντέλου που έχει εκπαιδευτεί πάνω σε εικόνες προσώπων από τη βάση δεδομένων MUCT. Η βάση αυτή περιέχει 3775 ποικίλες annotated εικόνες προσώπων (διαφορετικού φωτισμού, ηλικίας και εθνικότητας των ατόμων) στις οποίες έχουν προστεθεί 76 landmarks για την αναπαράσταση των χαρακτηριστικών του κάθε προσώπου. Η δημιουργία του μοντέλου βάσει αυτών των landmarks πραγματοποιείται χρησιμοποιώντας μια τεχνική γραμμικής μείωσης διαστάσεων των δεδομένων (Principal Component Analysis ή PCA) με σκοπό την ευκολότερη κατανόηση και επεξεργασία τους. Συνοπτικά, το PCA υπολογίζει τη μέση θέση των σημείων των landmarks κάθε εικόνας και εξάγει την απόκλιση τους σε κάθε πρόσωπο ως συνιστώσες διανυσμάτων. Το PCA εξάγει τις συνιστώσες αυτές με σειρά προτεραιότητας, συνήθως ξεκινώντας από βασικές αποκλίσεις όπως η κάθετη και οριζόντια στροφική κίνηση του προσώπου και συνεχίζοντας με αλλαγές όπως το άνοιγμα και κλείσιμο του στόματος, το χαμόγελο και άλλα. Στο παράδειγμα χρησιμοποιείται μια παραλλαγή του PCA, το Sparse PCA και το μοντέλο που προκύπτει αποθηκεύεται ως model_spca_20_svm.js.

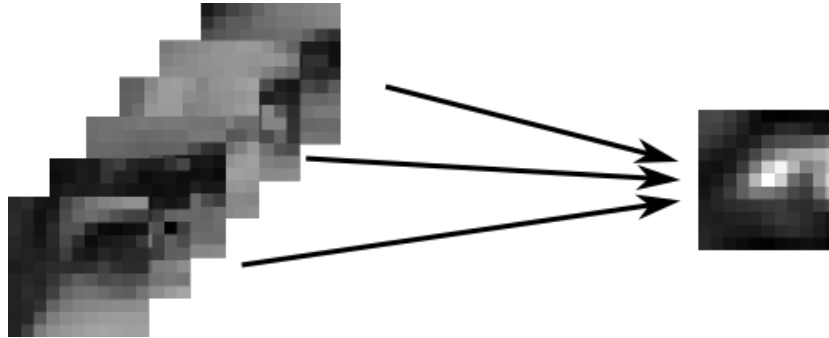


Εικόνα 2.1 (Αριστερά) το PCA facial model με τα 70 facial landmarks [7] και (δεξιά) μια annotated εικόνα από το MUCT database [8].

Ο αλγόριθμος χρησιμοποιεί 70 ταξινομητές (classifiers), ένα για κάθε σημείο, με τους οποίους πραγματοποιεί το fitting του μοντέλου. Η εκπαίδευση αυτών των classifiers, που αποσκοπούν στον όσο το δυνατό ακριβέστερο εντοπισμό των χαρακτηριστικών του προσώπου του χρήστη, γίνεται με τη χρήση patches. Για παράδειγμα, για να εκπαιδευτεί ο classifier του σημείου 27 εξάγεται μια μικρή τετραγωνική επιφάνεια (patch) X επί X για κάθε εικόνα του database που έχει ως επίκεντρο τη θέση του σημείου 27 στην εκάστοτε εικόνα ([Εικόνα 2.2](#)). Τα patches που δημιουργούνται χρησιμοποιούνται ως δεδομένα εισόδου για την εκπαίδευση του ταξινομητή, ο οποίος στην περίπτωση μας είναι ένας SVM classifier. Για να πραγματοποιηθεί το fitting του εκπαιδευμένου μοντέλου, η βιβλιοθήκη εξάγει μια περιοχή Y επί Y γύρω από κάθε αρχικό σημείο. Σε κάθε περιοχή, εφαρμόζει τον ανάλογο ταξινομητή ο οποίος παράγει ένα Y επί Y αποτέλεσμα (response) από pixels που αντιστοιχεί στην πιθανότητα η περιοχή αυτή να αναπαριστά το σωστό χαρακτηριστικό του προσώπου. Για κάθε response που παράγεται, το Clmtrackr υπολογίζει, με τον αλγόριθμο Mean-Shift⁸, το πώς χρειάζεται να τροποποιηθεί και να μετακινηθεί το μοντέλο, ώστε να ταιριάζει στο πρόσωπο που έχει εντοπίσει ([Εικόνα 2.3](#)). Στις νέες, περισσότερο ακριβείς, θέσεις των σημείων του μοντέλου εφαρμόζονται περιορισμοί, οι οποίοι εξασφαλίζουν την αρμονική μετακίνηση των σημείων σύμφωνα με το γενικό σχήμα

⁸ Στο παράδειγμα, ο Mean-Shift πραγματοποιεί visual tracking για να υπολογίσει την περισσότερο πιθανή (σωστή) περιοχή στην οποία πρέπει να μετακινηθεί το μοντέλο

του μοντέλου. Η διαδικασία αυτή πραγματοποιείται επαναληπτικά, μέχρι τα σημεία του μοντέλου να συμπίπτουν με τα χαρακτηριστικά του προσώπου.



Εικόνα 2.2 Εκπαίδευση ενός classifier από τις annotated εικόνες προσώπων [7].

Algorithm 1 Subspace Constrained Mean-Shifts

Require: I and p .

```

1: while not_converged( $p$ ) do
2:   Compute responses
3:   Linearize shape model
4:   Precompute pseudo-inverse of Jacobian ( $J^\dagger$ )
5:   Initialize parameter updates:  $\Delta p \leftarrow 0$ 
6:   while not_converged( $\Delta p$ ) do
7:     Compute mean-shifted landmarks
8:     Apply subspace constraint
9:   end while
10:  Update parameters:  $p \leftarrow p + \Delta p$ 
11: end while
12: return  $p$ 

```

Εικόνα 2.3 Ενδεικτικά τα βήματα του αλγόριθμου Mean-Shift σε ψευδογλώσσα [9].

Χρειάζεται να αναφερθεί πως κατά την αρχικοποίηση του μοντέλου, το Clmtrackr χρησιμοποιεί τη συνάρτηση faceDetection, η οποία αξιοποιώντας τις δυνατότητες της βιβλιοθήκης JSFeat⁹ βρίσκει μια εκτίμηση της θέσης του προσώπου και κατ' επέκταση των ματιών και της μύτης, οι οποίες χρησιμοποιούνται ως σημεία αναφοράς για την αρχική εφαρμογή του μοντέλου πάνω στο πρόσωπο. Καθ' όλη τη διαδικασία του face tracking η βιβλιοθήκη ελέγχει αν η θέση του μοντέλου έχει αποκλίνει από την πραγματική θέση του προσώπου, εξετάζοντας ανά τακτά χρονικά διαστήματα αν η περιοχή η οποία καλύπτει το

⁹ <https://github.com/inspirit/jsfeat>

μοντέλο παραπέμπει σε πρόσωπο. Σε περίπτωση που το παραπάνω δεν ισχύει πραγματοποιεί ξανά αρχικοποίηση της διαδικασίας της ανίχνευσης προσώπου.

2.9 Βιβλιοθήκη face_deformer

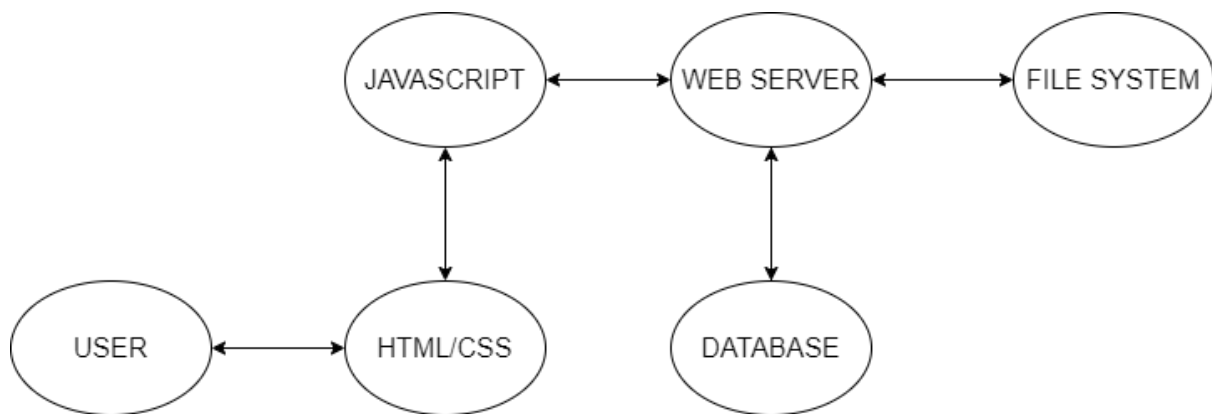
Εντός του index.html προστίθενται όλες οι απαραίτητες βιβλιοθήκες και το εκπαιδευμένο με τη μέθοδο SPCA μοντέλο που απαιτούνται για το face tracking και την εφαρμογή των προσώπων. Εκτός του Clmtrackr, στο παράδειγμά μας το οποίο πραγματοποιεί αλλαγή του προσώπου του χρήστη με το πρόσωπο από κάποια εικόνα, σημαντικό ρόλο έχει η βιβλιοθήκη face_deformer.js, η οποία είναι υπεύθυνη για την παραμόρφωση προσώπων είτε από εικόνα είτε από βίντεο και την εξαγωγή του αποτελέσματος σε ένα WebGL canvas. Στο script του index.html δημιουργούνται δύο μεταβλητές fd και fd2 στις οποίες ανατίθενται δύο instances από face deformer. Η αρχικοποίηση των face deformer γίνεται από τα δύο WebGL canvases webgl_overlay και webgl_overlay2. Ο δεύτερος deformer fd2 χρησιμοποιείται βοηθητικά και παραμορφώνει κατάλληλα το πρόσωπο της εικόνας που θέλουμε να εφαρμόσουμε ώστε να ταιριάζει με τις συντεταγμένες των landmarks του μοντέλου που έχει εφαρμοστεί στο πρόσωπο του χρήστη. Η διαδικασία αυτή γίνεται με τη συνάρτηση fd2.load() και το νέο παραμορφωμένο πρόσωπο αποτυπώνεται πάνω στο webgl_overlay2 με το fd2.draw(), το οποίο στη συνέχεια σχεδιάζεται πάνω στο newcanvas. Έπειτα, με τον πρώτο deformer φορτώνεται και εμφανίζεται στο χρήστη το τελικό αποτέλεσμα που ενώνει ομαλά με τη μέθοδο Poisson το πρόσωπο του newcanvas με το stream της κάμερας του χρήστη του videocanvas.



Εικόνα 2.4 Το annotated πρόσωπο μιας εικόνας, το πρόσωπο στην αρχική του θέση και το παραμορφωμένο πρόσωπο σε διαφορετική θέση.

3 Αρχιτεκτονική και Υλοποίηση της εφαρμογής

Όπως διακρίνεται και στο [Διάγραμμα 3.1](#) η web εφαρμογή αποτελείται από το frontend user interface με το οποίο επικοινωνεί ο χρήστης και περιλαμβάνει τα μέρη των HTML, CSS και JavaScript. Προκειμένου να εμφανιστούν τα δεδομένα που έχει ζητήσει ο χρήστης το frontend αλληλεπιδρά με το backend στέλνοντας requests και λαμβάνοντας responses. Εντός του backend ο web server ενημερώνει ή συλλέγει τα κατάλληλα δεδομένα επικοινωνώντας με τη βάση δεδομένων, καθώς και με το file system που περιλαμβάνει τα απαραίτητα HTML και JavaScript αρχεία και τις ανεβασμένες εικόνες του καταλόγου της εφαρμογής.



Διάγραμμα 3.1 Η αρχιτεκτονική της εφαρμογής

3.1 Διαδικασία του face substitution

Κατά το άνοιγμα της εφαρμογής ξεκινά η διαδικασία της φόρτωσης των масκών με τα πρόσωπα όλων των εικόνων που μπορούν να χρησιμοποιηθούν για το face substitution. Μόλις ολοκληρωθεί, καλείται η συνάρτηση `startVideo()` ([Κώδικας 3.1](#)) η οποία πραγματοποιεί το tracking του προσώπου με την εντολή `ctrack.start(vid)`. Εντός του `startVideo()` καλείται η συνάρτηση `drawGridLoop()` ([Κώδικας 3.2](#)) η οποία παίρνει με την εντολή `ctrack.getCurrentPosition()` τις θέσεις των σημείων του εντοπιζόμενου προσώπου και ελέγχει με το `ctrack.getConvergence()` αν το μοντέλο έχει εφαρμοστεί ορθά σε αυτό. Αν η τιμή του `getConvergence()` είναι μικρότερη από μια σταθερά που έχει θέσει ο δημιουργός (δηλαδή έχει γίνει σωστή εφαρμογή του μοντέλου) καλείται η συνάρτηση `switchMasks()` στην οποία δίνεται ως παράμετρος οι θέσεις των σημείων που έχουν επιστραφεί από το `ctrack`. Σε περίπτωση που

δεν πληρείται η παραπάνω προϋπόθεση καλείται ξανά το drawGridLoop() για το επόμενο animation frame μέχρι να επιτευχθεί μια ικανοποιητική τιμή.

```
function startVideo() {
    // set the webgl_overlay transparency to 1
    webgl_overlay.style.opacity = 1;
    // start tracking
    ctrack.start(vid);
    trackingStarted = true;
    // start drawing face grid
    drawGridLoop();

    // enable/disable the stop/start button
    document.getElementById('startbutton').disabled = true;
    document.getElementById('stopbutton').disabled = false;
}
```

Κώδικας 3.1 Η συνάρτηση startVideo().

```
function drawGridLoop() {
    // get position of face
    positions = ctrack.getCurrentPosition();

    overlayCC.clearRect(0, 0, vid_width, vid_height);
    // check whether mask has converged
    var pn = ctrack.getConvergence();
    if (pn < 0.4) {
        switchMasks(positions);
    } else {
        detectionRequest = requestAnimFrame(drawGridLoop);
    }
}
```

Κώδικας 3.2 Η συνάρτηση drawGridLoop().

Με τη switchMasks() ([Κώδικας 3.3](#)), αρχικά, σχεδιάζεται το vid (το οποίο περιέχει το stream της κάμερας) πάνω στο videocanvas με τη μέθοδο drawImage() του canvas API. Στη συνέχεια, γίνεται επεξεργασία του scale και του translation των σημείων του περιγράμματος των δύο προσώπων τροποποιώντας τις συντεταγμένες τους και αποθηκεύοντας τα στους πίνακες newMaskPos και newFacePos, ενώ προστίθενται επιπλέον vertices για την ενσωμάτωση επιπλέον περιοχών έξω από την αρχική μάσκα της εικόνας που θέλουμε να εφαρμοστεί. Το canvas της εικόνας που έχουμε επιλέξει, ο πίνακας newMaskPos με τις τροποποιημένες θέσεις των σημείων της εικόνας, το μοντέλο που κάνουμε fit στο πρόσωπο, καθώς και ο συνδυασμός των αρχικών και των επιπλέον vertices δίνονται ως παράμετροι στη συνάρτηση fd2.load() η οποία πραγματοποιεί τον εντοπισμό και την εξαγωγή του προσώπου από την εικόνα που

έχουμε επιλέξει να χρησιμοποιηθεί για το substitution. Μετά τη φόρτωση των προηγούμενων δεδομένων καλείται η `fd2.draw()` στην οποία δίνεται ως παράμετρος το `newFacePos` που ορίζει τις νέες θέσεις των χαρακτηριστικών πάνω στο πρόσωπο του χρήστη βάσει των οποίων θα εφαρμοστεί το deformed πρόσωπο. Τα αποτελέσματα του deformation τα οποία περιέχονται εντός του `webgl_overlay2` σχεδιάζονται στο `newcanvas` με τη μέθοδο `drawImage()`.

```
function switchMasks(pos) {
  // render result to videocanvas
  videocanvas.getContext('2d', { willReadFrequently: true }).drawImage(vid, 0, 0, videocanvas.width, videocanvas.height);
  videocanvas.imageSmoothingEnabled = true;

  // extend the positions with new estimated points in order to get pixels immediately outside mask
  var newMaskPos = masks[images[currentMask].id].slice(0);
  var newFacePos = pos.slice(0);
  var extInd = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 22, 21, 20, 19];
  var newwp;
  for (var i = 0; i < 23; i++) {
    newwp = [];
    newwp[0] = (newMaskPos[extInd[i]][0] * 1.3) - (newMaskPos[62][0] * 0.3);
    // short for ((newMaskPos[extInd[i]][0]-newMaskPos[62][0])*1.1)+newMaskPos[62][0]
    newwp[1] = (newMaskPos[extInd[i]][1] * 1.3) - (newMaskPos[62][1] * 0.3);
    newMaskPos.push(newwp);

    newwp = [];
    newwp[0] = (newFacePos[extInd[i]][0] * 1.3) - (newFacePos[62][0] * 0.3);
    newwp[1] = (newFacePos[extInd[i]][1] * 1.3) - (newFacePos[62][1] * 0.3);
    newFacePos.push(newwp);
  }
  // also need to make new vertices incorporating area outside mask
  var newVertices = pModel.path.vertices.concat(extended_vertices);

  // deform the mask we want to use to face from
  fd2.load(imageCanvases[images[currentMask].id], newMaskPos, pModel, newVertices);
  fd2.draw(newFacePos);
  // and copy onto new canvas
  newcanvas.getContext('2d', { willReadFrequently: true }).drawImage(webgl_overlay2, 0, 0);
  newcanvas.imageSmoothingEnabled = true;

  // create masking
  var tempcoords = positions.slice(0, 18);
  tempcoords.push(positions[21]);
  tempcoords.push(positions[20]);
  tempcoords.push(positions[19]);
  createMasking(maskcanvas, tempcoords);
}
```

Κώδικας 3.3 Η συνάρτηση switchMasks() (Μέρος 1).

Ακολουθεί η δημιουργία της μάσκας που διαχωρίζει το πρόσωπο κάθε εικόνας από φόντο. Εντός της `switchMasks()` δημιουργείται η μεταβλητή `tempcoords` η οποία παίρνει τις συντεταγμένες των 22 πρώτων σημείων που αντιστοιχούν στο περίγραμμα του προσώπου και καλεί τη συνάρτηση `createMasking()` ([Κώδικας 3.4](#)) στην οποία θέτει ως παραμέτρους το `maskcanvas` (για τη σχεδίαση της μάσκας) και το `tempcoords`. Στην `createMasking()` δημιουργείται η μεταβλητή `cc` η οποία, χρησιμοποιώντας τη μέθοδο `getContext()`, μετατρέπεται σε reference στο `maskcanvas` με αποτέλεσμα όσες αλλαγές πραγματοποιηθούν πάνω σε αυτή να μεταφερθούν και στο `maskcanvas`. Συγκεκριμένα, χρησιμοποιώντας μεθόδους και ιδιότητες του canvas API η συνάρτηση γεμίζει με μαύρο χρώμα ολόκληρο το

maskcanvas και στη συνέχεια δημιουργεί ένα path, χρησιμοποιώντας τα σημεία που δόθηκαν από τη μεταβλητή tempcoords, γύρω από το κεφάλι το οποίο γεμίζει με άσπρο χρώμα. Με τη διαδικασία αυτή, επιτυγχάνεται ο διαχωρισμός του προσώπου από την υπόλοιπη εικόνα προκειμένου να χρησιμοποιηθεί στο Poisson Blending.

```
function createMasking(canvas, modelpoints) {  
  // fill canvas with black  
  var cc = canvas.getContext('2d', { willReadFrequently: true });  
  cc.imageSmoothingEnabled = true;  
  cc.fillStyle = "#000000";  
  cc.fillRect(0, 0, canvas.width, canvas.height);  
  // create a path around the positions of the face and fill it with white  
  cc.beginPath();  
  cc.moveTo(modelpoints[0][0], modelpoints[0][1]);  
  for (var i = 1; i < modelpoints.length; i++) {  
    cc.lineTo(modelpoints[i][0], modelpoints[i][1]);  
  }  
  cc.lineTo(modelpoints[0][0], modelpoints[0][1]);  
  cc.closePath();  
  cc.fillStyle = "#ffffff";  
  cc.fill();  
}
```

Κώδικας 3.4 Η συνάρτηση createMasking().

Επιστρέφοντας στη switchMasks() ([Κώδικας 3.5](#)), μετά από την κλήση του createMasking(), καλείται η συνάρτηση Poisson.load() μέσω της βιβλιοθήκης poisson_new.js. Η συνάρτηση αυτή δέχεται ως ορίσματα το newcanvas (που περιέχει το νέο πρόσωπο), το videocanvas (που περιέχει το stream του χρήστη) και, τέλος το maskcanvas (που περιέχει τη μάσκα του νέου προσώπου). Μόλις η φόρτωση τους ολοκληρωθεί, καλεί την Poisson.blend() η οποία επιστρέφει τον πίνακα των pixels με το χρωματικά διορθωμένο (με βάση τα χρώματα του stream) πρόσωπο της εικόνας και το εφαρμόζει πάνω στο newcanvas με τη μέθοδο putImageData(). Στο τελικό κομμάτι της η switchMasks() καλεί την fd.load() η οποία φορτώνει το επεξεργασμένο και τελικό newcanvas στον deformer για την παραμόρφωσή του και έπειτα καλεί επαναληπτικά για κάθε animation frame τη συνάρτηση drawMaskLoop() με τη μέθοδο requestAnimationFrame().

Η drawMaskLoop() ([Κώδικας 3.6](#)) αποτελεί το τελευταίο βήμα που απαιτείται για τη διαδικασία του face substitution. Μέσω αυτής αποκτώνται ξανά οι τρέχουσες θέσεις των σημείων του προσώπου του χρήστη με το ctrack.getCurrentPosition() και όταν βρεθούν καλείται η συνάρτηση fd.draw() η οποία σχεδιάζει το τελικό παραμορφωμένο πρόσωπο πάνω στο newcanvas. Τέλος, η drawMaskLoop() καλεί τον εαυτό της εντός του

requestAnimationFrame() επιτυγχάνοντας ένα συνεχές ομαλό tracking του προσώπου για κάθε animation frame.

```
// do poisson blending
Poisson.load(newcanvas, videocanvas, maskcanvas, function() {
    var result = Poisson.blend(30, 0, 0);
    // render to canvas
    newcanvas.getContext('2d', { willReadFrequently: true }).putImageData(result, 0, 0);
    newcanvas.imageSmoothingEnabled = true;

    // final deformation
    fd.load(newcanvas, pos, pModel);
    animationRequest = requestAnimFrame(drawMaskLoop);
});
}
```

Κώδικας 3.5 Η συνάρτηση switchMasks() (Μέρος 2).

```
function drawMaskLoop() {
    animationRequest = requestAnimFrame(drawMaskLoop);
    // get position of face
    positions = ctrack.getCurrentPosition();

    overlayCC.clearRect(0, 0, vid_width, vid_height);
    if (positions) {
        // draw mask on top of face
        fd.draw(positions);
    }
}
```

Κώδικας 3.6 Η συνάρτηση drawMaskLoop().

3.2 Επιπλέον λειτουργίες της εφαρμογής

Ο δημιουργός παρέχει στη σελίδα της βιβλιοθήκης στο GitHub έτοιμες HTML σελίδες με παραδείγματα χρήσης της. Για την εργασία χρησιμοποιήθηκε το αρχείο facesubstitution.html το οποίο μετονομάστηκε σε index.html και είναι υπεύθυνο για τη διαδικασία της αλλαγής του προσώπου. Πάνω σε αυτό προστέθηκαν επιπλέον λειτουργίες για μια περισσότερο ολοκληρωμένη και ομαλή εμπειρία για το χρήστη. Καθώς η χρήση του getUserMedia() απαιτεί την εκτέλεση του κώδικα μέσω server ή localhost (και όχι απευθείας από το αρχείο) όλα τα αρχεία τρέχουν μέσω του Node.js. Αρχικά χρησιμοποιήθηκε το module http-server το οποίο κάνει serve στατικά HTML αρχεία στο localhost, όμως επειδή θα χρειάζεται να μπορούμε να μεταφερόμαστε σε διαφορετικές σελίδες μέσα στην εφαρμογή επιλέχθηκε διαφορετική

προσέγγιση. Έτσι, δημιουργήθηκε ένα ξεχωριστό server.js αρχείο το οποίο εκτελείται με την εντολή node server.js και δημιουργεί το localhost στον υπολογιστή του χρήστη.

Για τη λειτουργία του, το server.js χρησιμοποιεί modules ([Κώδικας 3.7](#)) που παρέχει το Node τα οποία εγκαθίστανται στο φάκελο της εφαρμογής με την εντολή npm install. Υπεύθυνο για την αρχικοποίηση και το σχεδιασμό της web εφαρμογής είναι το express.js module το οποίο αποτελεί ένα framework που απλοποιεί τη διαδικασία της δημιουργίας σταθερών και επεκτάσιμων διαδικτυακών εφαρμογών. Στο server.js δημιουργείται instance του express στη μεταβλητή app μέσω της οποίας καθορίζονται η αρχική σελίδα που εμφανίζεται στο χρήστη, καθώς και ο φάκελος από τον οποίο θα κάνει serve τα αρχεία. Μέσω της μεταβλητής app πραγματοποιούνται όλες οι server-side λειτουργίες για τις οποίες θα μιλήσουμε στη συνέχεια. Τέλος, γίνεται η εκκίνηση του server με την εντολή app.listen() στο port το οποίο έχουμε καθορίσει (π.χ. 8080). Παράλληλα με το express χρησιμοποιείται και το express-session το οποίο δημιουργεί sessions για κάθε συνδεδεμένο χρήστη και αποθηκεύει σημαντικά δεδομένα (όπως το id του χρήστη) εντός του server.

```
// declaration of required modules
const express = require('express');
const session = require('express-session');
const path = require('path');
const multer = require('multer');
const bcrypt = require('bcrypt');
const bodyParser = require('body-parser');

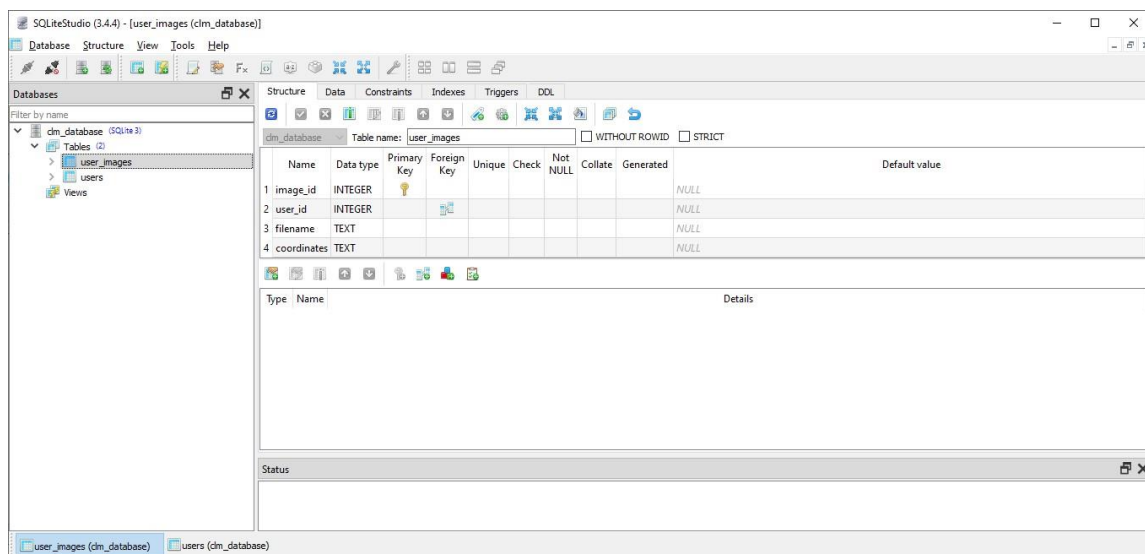
// SQLite initialization
const sqlite3 = require('sqlite3').verbose();

// create a directory to store uploaded images
const uploadDirectory = './static/uploads';
const storage = multer.diskStorage({
  destination: uploadDirectory,
  filename: (req, file, callback) => {
    callback(null, file.originalname); // use the original file name
  }
});
const upload = multer({ storage });
```

Κώδικας 3.7 Δήλωση των απαραίτητων modules και δημιουργία του directory για το ανέβασμα των εικόνων του χρήστη.

Καθώς στόχος είναι η δημιουργία μιας εξατομικευμένης εμπειρίας για το χρήστη προστέθηκε η λειτουργία δημιουργίας λογαριασμού ο οποίος απαιτείται για τη σύνδεσή του στην εφαρμογή. Η λειτουργία αυτή κρίθηκε απαραίτητη διότι θέλαμε να δώσουμε στο χρήστη τη δυνατότητα προσθήκης δικών του εικόνων με πρόσωπα τα οποία μπορούν να

χρησιμοποιηθούν για το face substitution. Έτσι, δημιουργήθηκε με το SQLite το αρχείο `clm_database.db` το οποίο αποτελεί τη βάση δεδομένων της εφαρμογής. Το SQLite ([Εικόνα 3.1](#)) αποτελεί μια μηχανή βάσης δεδομένων που χρησιμοποιείται ευρέως από εφαρμογές σε προγράμματα περιήγησης και στο λειτουργικό σύστημα, καθώς σε αντίθεση με τα κλασικά συστήματα βάσεων δεδομένων δεν απαιτεί την ύπαρξη database administrator για την εκκίνηση της βάσης σε ξεχωριστό περιβάλλον. Το SQLite λειτουργεί ως βιβλιοθήκη η οποία συνδέεται απευθείας στην εφαρμογή και επιτρέπει σε αυτήν την επεξεργασία της βάσης. Παράλληλα διαθέτει και το SQLiteStudio, ένα περιβάλλον που διευκολύνει τη διαχείριση της βάσης δεδομένων. Η ενσωμάτωσή του στο `server.js` γίνεται με το `sqlite3` module μέσω του οποίου δημιουργείται ένα instance του `clm_database.db` στη μεταβλητή `db`. Στο πρόγραμμα του SQLite δημιουργούνται οι πίνακες `users` και `user_images` οι οποίοι περιέχουν τα στοιχεία των χρηστών και των εικόνων που έχουν ανεβάσει αντίστοιχα. Ο πίνακας `users` περιέχει τις στήλες `user_id`, `username` και `password` με το `user_id` να αποτελεί το πρωτεύον κλειδί, ενώ ο `user_images` αποτελείται από τις στήλες `image_id` (πρωτεύον κλειδί), `user_id` (ξένο κλειδί που συνδέει τους δύο πίνακες), `filename` (όνομα του αρχείου) και `coordinates` (για την αποθήκευση των συντεταγμένων των facial landmarks του προσώπου).



Εικόνα 3.1 Η βάση δεδομένων στο περιβάλλον του SQLiteStudio.

Για την εισαγωγή των δεδομένων του εκάστοτε χρήστη στο database χρειάζεται να μεσολαβήσει ο server. Η διαδικασία αυτή γίνεται με τη χρήση των HTTP routes του express και συγκεκριμένα με τη μέθοδο `app.post()`. Η μέθοδος `app.post()` ορίζεται από ένα μονοπάτι `path` και χρησιμοποιείται για τη δημιουργία callback συναρτήσεων οι οποίες καλούνται όποτε

εμφανίζεται ένα HTTP POST request προς το path αυτό. Τα πρώτα δύο routes εξυπηρετούν το login και το register και ορίζονται από τα μονοπάτια /login και /register αντίστοιχα. Στην αρχική σελίδα, login_register.html, που εμφανίζεται στο χρήστη πραγματοποιούνται η εγγραφή και σύνδεση του στην εφαρμογή με τη χρήση forms. Το αρχικό form πραγματοποιεί το login του χρήστη και μόλις υποβληθεί στέλνει τα δεδομένα στο /login route, ενώ αν ο χρήστης επιλέξει να κάνει εγγραφή εμφανίζεται το form του register το οποίο με την υποβολή του στέλνει τα δεδομένα στο /register route. Εντός του /register route ([Κώδικας 3.8](#)) στο αρχείο server.js, ανακτώνται τα στοιχεία που έχει υποβάλει ο χρήστης και πραγματοποιείται σύνδεση στη βάση δεδομένων, στην οποία καταχωρούνται το username και το κρυπτογραφημένο (με τη χρήση του bcrypt module) password. Εάν η διαδικασία της εγγραφής είναι επιτυχής γίνεται ανακατεύθυνση στη σελίδα register_success. Παρόμοια, το /login route ([Κώδικας 3.9](#)) δημιουργεί σύνδεση προς τη βάση με την οποία ελέγχει αν τα στοιχεία (username και αποκρυπτογραφημένο password) που έχουν δοθεί υπάρχουν εντός του database. Σε περίπτωση που το παραπάνω ισχύει ο χρήστης ανακατευθύνεται στη βασική σελίδα του face substitution, ενώ αν κάποιο από τα στοιχεία είναι λανθασμένο εμφανίζεται η σελίδα failed_login.html.

```

// define a route to handle the form submission
app.post('/register', (req, res) => {
  const { username, password } = req.body;
  // open a connection to the SQLite database
  const db = new sqlite3.Database('clm_database.db');

  // check if the username exists in the database
  const sql = 'SELECT * FROM users WHERE username = ?';
  db.get(sql, [username], (err, row) => {
    if (err) {
      // handle the error
      res.status(500).json({ error: 'Server error' });
    }
    else {
      // username is available, proceed with registration
      // hash and store the password, insert the new user into the database, etc.

      bcrypt.hash(password, saltRounds, function(err, hash) {

        // insert the user into the database
        const sql = 'INSERT INTO users (username, password) VALUES (?, ?)';
        db.run(sql, [username, hash], (err) => {
          if (err) {
            console.error(err.message);
            res.send('Registration failed.');
```

Κώδικας 3.8 Το register route, το οποίο αναλαμβάνει την εγγραφή κάθε νέου χρήστη και την εισαγωγή των στοιχείων του στη βάση δεδομένων.

```

// define a route to handle login form submissions
app.post('/login', (req, res) => {

  const { username, password } = req.body;

  // open a connection to the SQLite database
  const db = new sqlite3.Database('clm_database.db');

  // retrieve the hashed password from the database based on the provided username
  const sql = 'SELECT * FROM users WHERE username = ?';
  db.get(sql, [username], (err, row) => {
    if (err) {
      console.error(err.message);
      res.send('Login failed.');
```

```

    } else if (row) {
      const hashedPasswordFromDatabase = row.password;

      // compare the submitted password with the stored hash
      bcrypt.compare(password, hashedPasswordFromDatabase, function(err, result) {
        if (result) {
          // Passwords match, login successful
          const userId = row.user_id; // Retrieve the user ID from the database row
          console.log('User ID:', userId);

          // you can set the user's ID in the session or handle it as needed here
          req.session.userId = userId;

          // redirect to a secured page
          res.redirect('/html/index.html');
```

```

        } else {
          // Passwords do not match, login failed
          res.redirect('/html/failed_login.html');
```

```

        }
        db.close();
      });
    } else {
      // user not found
      res.redirect('/html/failed_login.html');
```

```

      db.close();
    }
  });
});
});

```

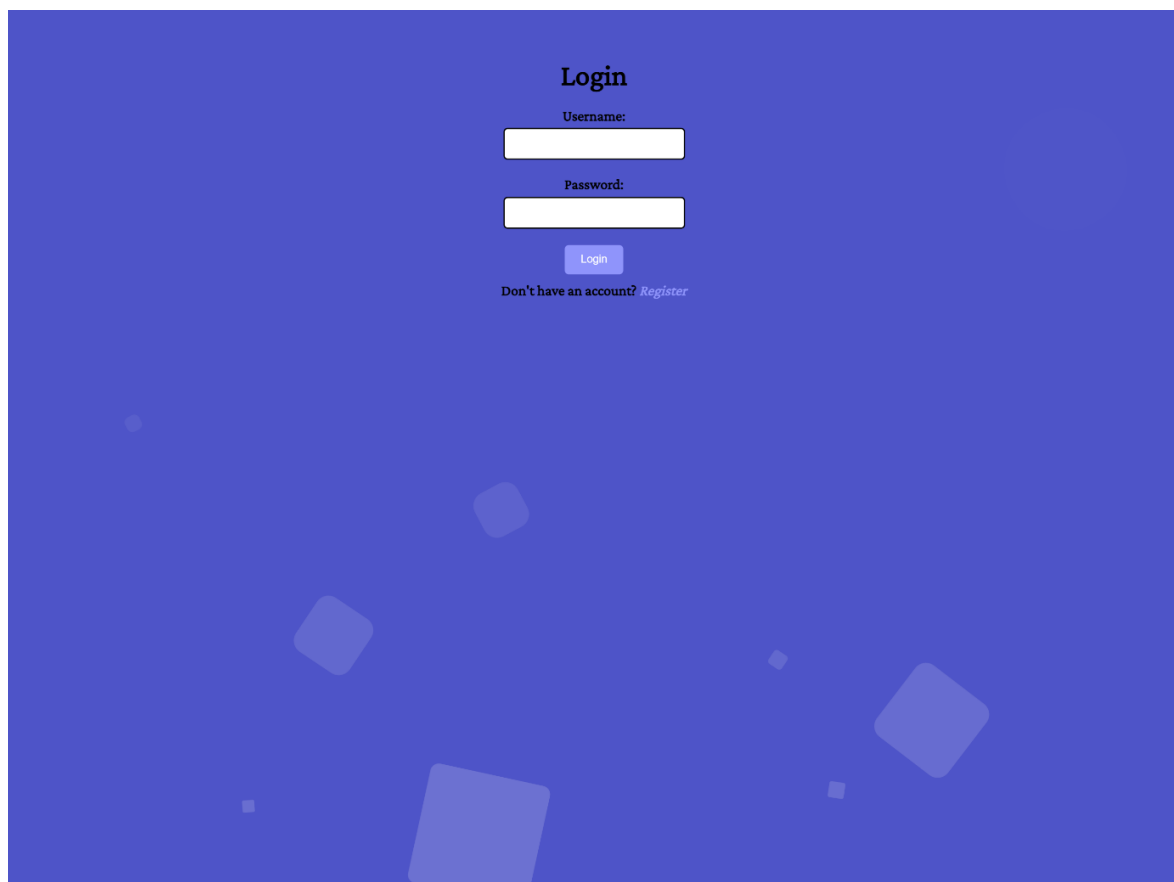
Κώδικας 3.9 Το login route, το οποίο ανακατευθύνει κάθε συνδεδεμένο χρήστη στη σελίδα index της εφαρμογής.

3.3 Frontend User Interface

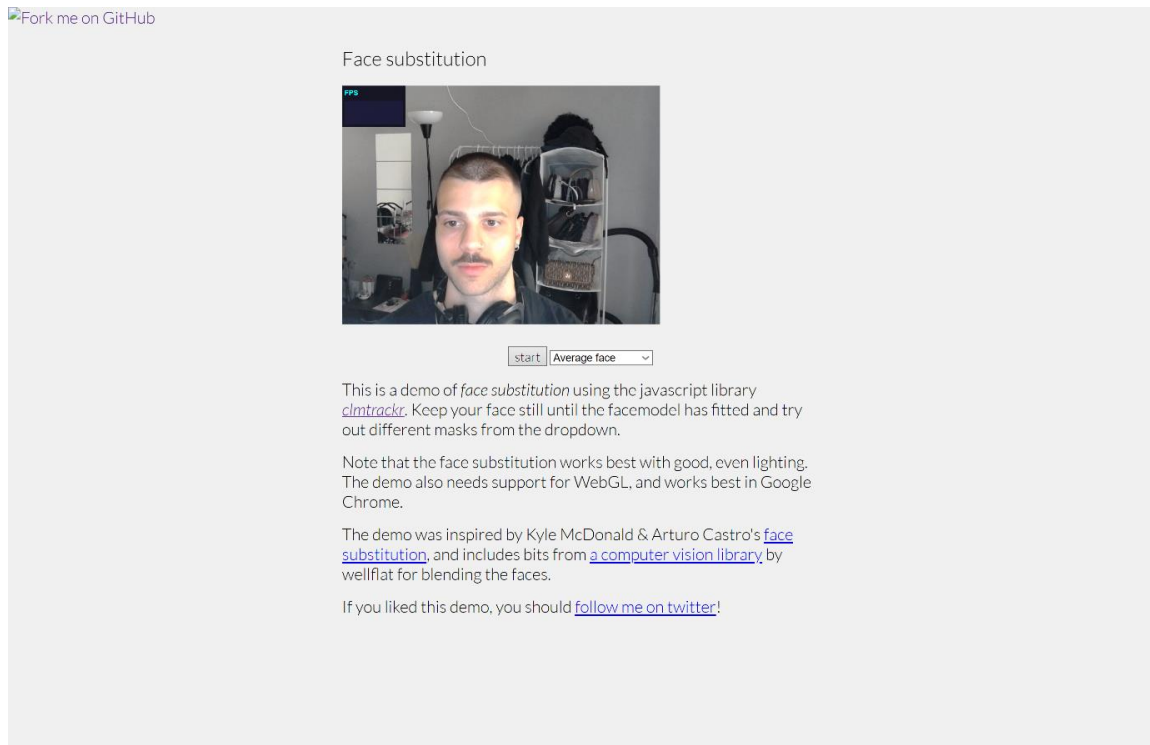
Η εφαρμογή αποτελείται από 3 HTML σελίδες, τη login/register.html για την είσοδο και εγγραφή του χρήστη, το index.html που περιέχει την καταγραφή βίντεο και τη βασική λειτουργία του face substitution και το annotator.html για το tracking των ανεβασμένων εικόνων. Για το login/register χρησιμοποιήθηκαν τα HTML forms για τη εισαγωγή και συλλογή των στοιχείων του χρήστη και προστέθηκε animated background¹⁰ σε CSS για τη

¹⁰ <https://codepen.io/mohaiman/pen/MQqMyo>

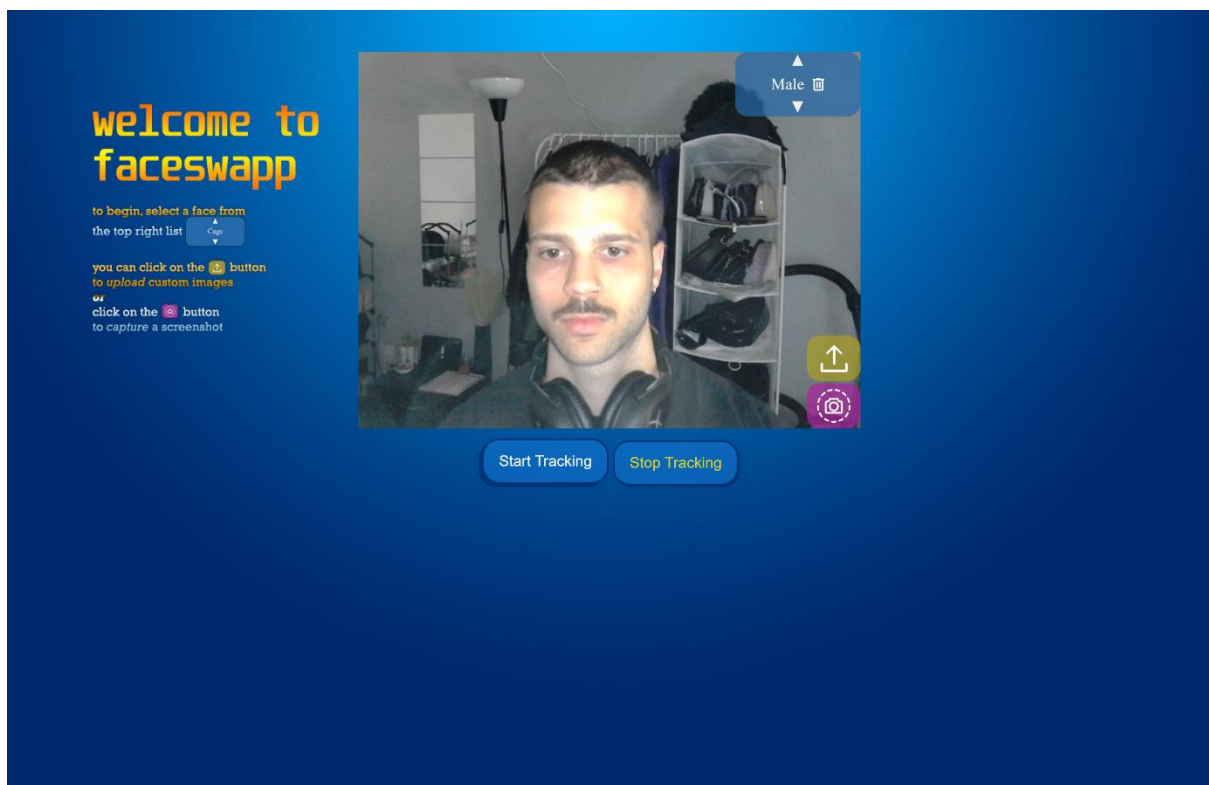
βελτίωση της εμφάνισης. Γενικότερα, και στις 3 σελίδες έγινε συνδυασμός HTML, CSS και JavaScript για τη βελτίωση της αισθητικής και την επίτευξη συνοχής κατά την πλοήγησή τους. Το index.html αποτελείται από το παράθυρο του video stream της κάμερας του χρήστη στο οποίο εφαρμόζονται τα επιπλέον canvases που πραγματοποιούν την επεξεργασία του προσώπου. Στη γωνία του stream υπάρχει ένα διαδραστικό container το οποίο περιέχει τα ονόματα των διαθέσιμων εικόνων που μπορούν να χρησιμοποιηθούν για το face substitution και μεγαλώνει όταν ο χρήστης πατήσει πάνω του. Τις διαθέσιμες επιλογές τις παίρνει από το selectmask element το οποίο χρησιμοποιείται στο αρχικό face_substitution.html και έχει τη μορφή drop-down λίστας ([Εικόνα 3.3](#)). Στην τελική μορφή του face substitution έχει γίνει απόκρυψη του selectmask και χρησιμοποιείται μόνο βοηθητικά για τη φόρτωση του container με τις διαθέσιμες εικόνες και τη δυνατότητα του χρήστη να αλλάζει το πρόσωπο που εφαρμόζεται ([Εικόνα 3.5](#)). Κάτω από το container υπάρχουν δύο κουμπιά για τη μεταφόρτωση εικόνων και τη λήψη φωτογραφιών του stream, ενώ κάτω από το παράθυρο του stream υπάρχουν ακόμα δύο κουμπιά που δίνουν τη δυνατότητα στο χρήστη να ξεκινήσει ή να διακόψει τη διαδικασία του tracking. Σχετικά με το annotator.html γίνεται αναφορά στην ενότητα 3.5.



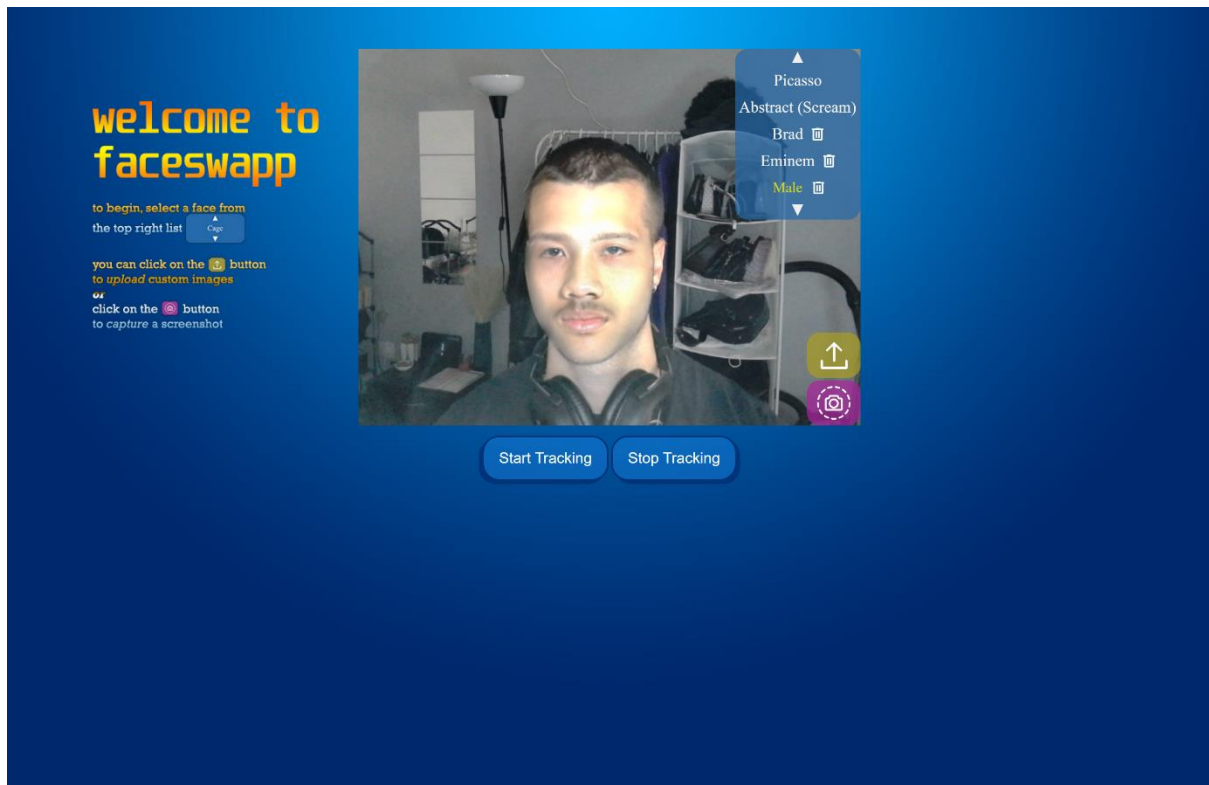
Εικόνα 3.2 Η σελίδα σύνδεσης και εγγραφής του χρήστη.



Εικόνα 3.3 Το αρχικό περιβάλλον της σελίδας του face substitution με σταματημένο το tracking.



Εικόνα 3.4 Το τελικό περιβάλλον της σελίδας του face substitution με σταματημένο το tracking.



Εικόνα 3.5 Το τελικό περιβάλλον της σελίδας του face substitution με ανοιγμένο το container και επιλεγμένο το πρόσωπο Male το οποίο έχει ανεβάσει ο χρήστης.

3.4 Upload εικόνας

Για το upload γίνεται ξανά χρήση ενός form το οποίο στέλνει το ανεβασμένο αρχείο στο server μέσω του /upload route ([Κώδικας 3.10](#)). Αυτή τη φορά, το `app.post()` του upload καλεί και μια επιπλέον συνάρτηση την `upload.single()`. Η συνάρτηση αυτή αποτελεί μέρος του `multer` module το οποίο χρησιμοποιείται ευρέως για τη μεταφόρτωση αρχείων με το `express`. Έτσι, εντός του server δημιουργείται η μεταβλητή `upload` στην οποία αρχικοποιούμε το `multer` αφού πρώτα έχουμε καθορίσει το φάκελο στον κατάλογο της εφαρμογής στον οποίο θέλουμε να ανέβουν οι εικόνες. Καλώντας, λοιπόν, το `upload.single()` (στο οποίο δίνουμε ως παράμετρο την εικόνα που επέλεξε ο χρήστης στο `index.html`) εξασφαλίζεται το ανέβασμα του αρχείου στον κατάλογο της εφαρμογής. Παράλληλα, μέσα στο `app.post()` γίνεται εισαγωγή του ονόματος της εικόνας στη βάση δεδομένων δίνοντας ως `user_id` το `id` του τρέχοντος χρήστη από το `express-session`. Αφού η μεταφόρτωση και η καταχώρηση είναι επιτυχής ο server ανακατευθύνει το χρήστη στη σελίδα του `annotator.html` μεταφέροντας το όνομα της εικόνας ως παράμετρο URL.

```

// define a route to handle image uploads
app.post('/upload', upload.single('image'), (req, res) => {
  var { originalname } = req.file;

  // check if the user is logged in
  if (req.session.userId) {
    const db = new sqlite3.Database('clm_database.db');

    // check if the filename already exists for the specific user
    const sqlCheck = 'SELECT COUNT(*) AS count FROM user_images WHERE user_id = ? AND filename = ?';

    db.get(sqlCheck, [req.session.userId, originalname], (checkErr, row) => {
      if (checkErr) {
        console.error(checkErr.message);
        res.send('Image upload failed.');
```

Κώδικας 3.10 Το upload route, το οποίο εισάγει τα δεδομένα στη βάση και ανακατευθύνει το χρήστη στο περιβάλλον του annotator.

3.5 Επεξεργασία της ανεβασμένης εικόνας

Η σελίδα annotator.html περιλαμβάνεται στο φάκελο με τα εργαλεία του Clmtrackr στο GitHub του Clmtrackr. Για τις λειτουργίες της χρησιμοποιεί ξεχωριστό script μέσα από το αρχείο annotator.js. Στην αρχική της μορφή αποτελεί ένα εξωτερικό βοηθητικό εργαλείο με το οποίο ο χρήστης ανεβάζει μια εικόνα και αυτόματα πραγματοποιείται εφαρμογή του μοντέλου SPCA στο εικονιζόμενο πρόσωπο μέσω του Clmtrackr ([Εικόνα 3.6](#)). Εάν τα χαρακτηριστικά του προσώπου δεν εμφανίζονται καθαρά και ο εντοπισμός δεν είναι ακριβής ο χρήστης μπορεί να μετακινήσει χειροκίνητα κάθε σημείο στο χαρακτηριστικό του προσώπου που αναλογεί. Η εξαγωγή των συντεταγμένων των θέσεων των facial landmarks γίνεται με το κουμπί export

current coordinates και ο χρήστης πρέπει να τις εισάγει χειροκίνητα στον πίνακα masks στο index.html για να μπορούν να χρησιμοποιηθούν στο face substitution. Το annotator.html δέχθηκε αρκετές τροποποιήσεις προκειμένου να συμβαδίζει με τις ανάγκες της εφαρμογής με αποτέλεσμα την τροποποίηση ή αφαίρεση περιττών λειτουργιών και κουμπιών, όπως η εξαγωγή συντεταγμένων, η δυνατότητα μεγέθυνσης της εικόνας, η αποθήκευση των annotations σε csv και άλλα ([Εικόνα 3.7](#)). Συνεπώς, κατά τη φόρτωση της σελίδας γίνεται εξαγωγή της παραμέτρου URL (δηλαδή του ονόματος της εικόνας) που μεταβιβάστηκε από το server και δίνεται (μαζί με το path της) ως παράμετρος στη συνάρτηση addCustomImageToFiles() η οποία βρίσκεται στο annotator.js ([Κώδικας 3.11](#)). Η συνάρτηση χρησιμοποιεί το XMLHttpRequest API με το οποίο πραγματοποιεί ένα ασύγχρονο request που κάνει fetch την εικόνα σε μορφή blob. Στη συνέχεια, δημιουργεί ένα αντικείμενο File μέσω της fetched εικόνας το οποίο δίνει ως παράμετρο στη συνάρτηση handleFileSelect(), η οποία εμφανίζει την εικόνα στο canvas της σελίδας και ξεκινά τη διαδικασία εντοπισμού του προσώπου. Αφού ο εντοπισμός έχει ολοκληρωθεί (και ο χρήστης είναι ικανοποιημένος με τη θέση κάθε σημείου του μοντέλου) δίνεται η δυνατότητα αποδοχής ή απόρριψης των αλλαγών. Εάν ο χρήστης πατήσει το κουμπί απόρριψης καλείται η συνάρτηση discardChanges() η οποία στέλνει μέσω του /discardUpload route το όνομα της εικόνας στο server ([Κώδικας 3.12](#)). Εντός του app.get('/discardUpload') διαγράφεται η καταχωρημένη εικόνα από τη βάση δεδομένων για το συγκεκριμένο χρήστη και πραγματοποιείται ανακατεύθυνση του στο index.html. Εάν γίνει αποδοχή των αλλαγών, καλείται η συνάρτηση redirectFunction() του annotator.js η οποία μεταφέρει μέσω URL τις συντεταγμένες των τελικών landmarks και το όνομα της εικόνας στο /setCoordinates route του server. Το app.get('/setCoordinates') ενημερώνει τη στήλη coordinates της εικόνας στη βάση δεδομένων με τον ανακτημένο πίνακα των συντεταγμένων και ανακατευθύνει το χρήστη πίσω στο index.html, όπου πλέον μπορεί να χρησιμοποιήσει το πρόσωπο της ανεβασμένης εικόνας στο face substitution.

```

// create a function to add custom images to the fileList
function addCustomImageToFiles(imagePath) {
  // create a new XMLHttpRequest to fetch the image data
  const xhr = new XMLHttpRequest();

  // set up the request to fetch the image as a blob
  xhr.open('GET', imagePath, true);
  xhr.responseType = 'blob';

  xhr.onload = function () {
    if (xhr.status === 200) {
      // create a File object from the fetched blob
      const customImage = new File([xhr.response], 'custom_image.png', { type: 'image/png' });

      // trigger the handleFileSelect function to load the custom image
      handleFileSelect(customImage);
    } else {
      console.error('Failed to fetch custom image:', xhr.status);
    }
  };

  xhr.send();
}

```

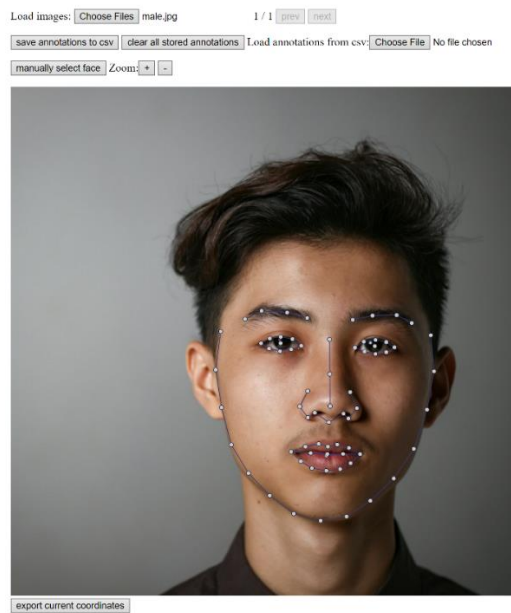
Κώδικας 3.11 Η συνάρτηση addCustomImageToFiles() εντός του annotator.js.

```

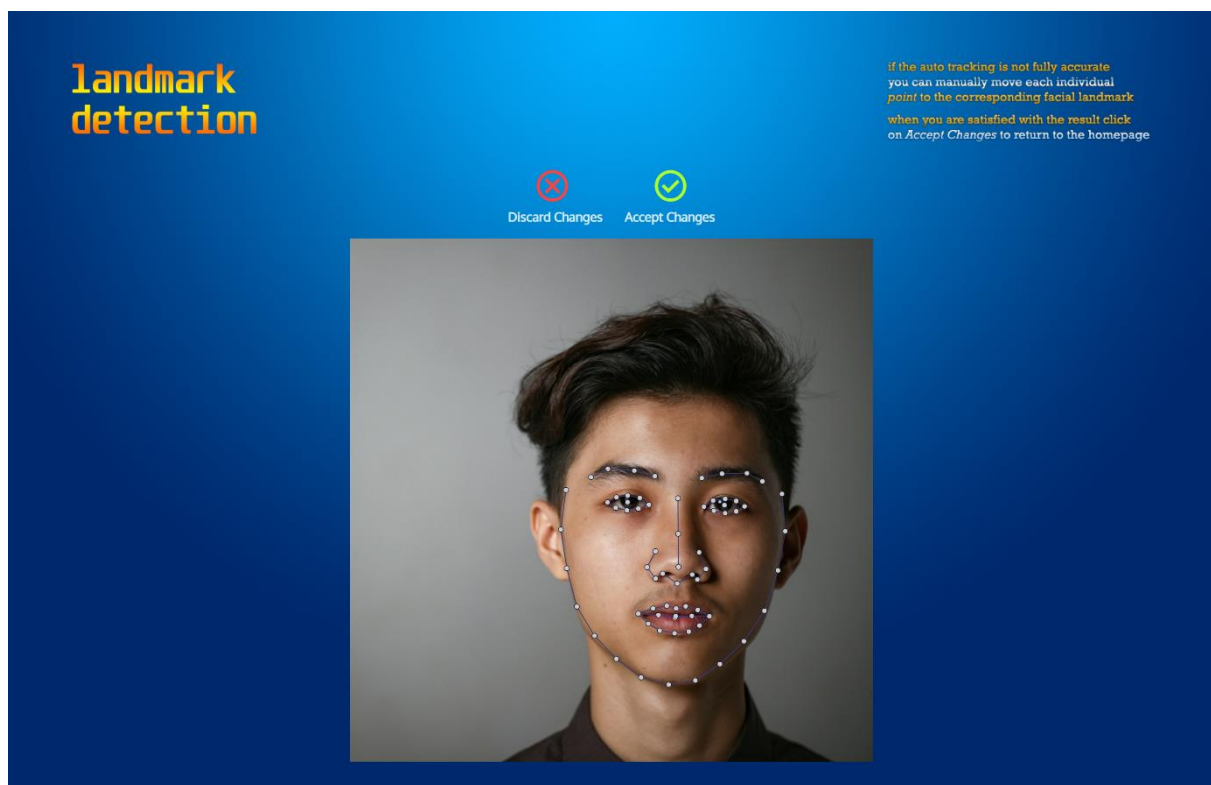
// define a route to discard the upload of an image in annotator.html
// (also used for deleting already uploaded images in client.js and annotator.html)
app.get('/discardUpload', (req, res) => {
  const originalname = req.query.image_id;
  if (req.session.userId) {
    const db = new sqlite3.Database('clm_database.db');

    const sql = 'DELETE FROM user_images WHERE user_id = ? AND filename = ?';
    db.run(sql, [req.session.userId, originalname], (err) => {
      if (err) {
        console.error(err.message);
        res.send('Discard upload failed.');
```

Κώδικας 3.12 Το discardUpload route εντός του server.



Εικόνα 3.6 Η αρχική σελίδα του annotator.html με το tracked πρόσωπο της ανεβασμένης εικόνας.



Εικόνα 3.7 Η τροποποιημένη σελίδα του annotator.html με το tracked πρόσωπο της ανεβασμένης εικόνας.

3.6 Client.js script

Το βασικότερο στοιχείο που εμπλουτίζει και ενώνει το index.html με τα δεδομένα που λαμβάνει ο server από το database είναι το script που βρίσκεται στο εξωτερικό αρχείο client.js. Το script αυτό παίρνει τα στοιχεία των εικόνων που έχει ανεβάσει ο χρήστης και αφού τα τροποποιήσει κατάλληλα τα μεταφέρει στο index.html για να χρησιμοποιηθούν από το χρήστη. Παράλληλα, είναι υπεύθυνο για τις λειτουργίες του container που παρουσιάζει στο χρήστη τα ονόματα των εικόνων.

3.7 Δεδομένα του getSelectData route

Για τη λήψη των ονομάτων των ανεβασμένων εικόνων το script χρησιμοποιεί τη συνάρτηση updateSelectWithData() η οποία λειτουργεί ασύγχρονα και στέλνει HTTP GET request στο /getSelectData route. Στο server.js εντός του app.get('/getSelectData') πραγματοποιείται σύνδεση στη βάση και γίνεται ανάκτηση του filename όλων των καταχωρημένων εικόνων για το συνδεδεμένο χρήστη ([Κώδικας 3.13](#)). Εάν βρεθούν ανεβασμένες εικόνες γίνεται διαχωρισμός του ονόματός τους από την επέκταση (JPG ή PNG) με τη μέθοδο map() και στη συνέχεια δημιουργείται το object responseData στο οποίο προστίθενται οι πίνακες του καθαρού ονόματος και του ονόματος με την επέκταση. Το responseData στέλνεται, στη συνέχεια, ως response σε μορφή JSON. Πίσω στην updateSelectWithData() του client.js δημιουργείται ένα fetch request στο /getSelectData μέσω του οποίου λαμβάνονται τα δύο στοιχεία σε μορφή JSON ([Κώδικας 3.14](#)). Τέλος, η συνάρτηση προσθέτει τις επιπλέον επιλογές στο selectmask element του index.html (το οποίο περιέχει τη λίστα με τις ήδη υπάρχουσες εικόνες), δίνοντας ως attribute σε κάθε επιλογή το πλήρες όνομα της μαζί με την επέκταση.

```

// define a route to fetch the images' names from the database (called in client.js)
app.get('/getSelectData', (req, res) => {
  const db = new sqlite3.Database('clm_database.db');
  // fetch data from the database
  userId = req.session.userId;
  const sql = 'SELECT filename FROM user_images WHERE user_id = ?';
  db.all(sql, [req.session.userId], (err, rows) => {
    if (err) {
      console.error('Error fetching custom images:', err);
      res.status(500).send('Internal Server Error');
    }

    else if (rows && rows.length > 0) {

      // extracting the filename without the extension which will be displayed in index.html as an option
      const customImagesNames = rows.map((row) => path.parse(row.filename).name);
      // extracting the full filename with the extension which will later be set as an attribute for each image
      const extensions = rows.map((row) => row.filename);

      // create an object with both arrays
      const responseData = {
        selectData: customImagesNames,
        imageExtensions: extensions
      };

      // send the data as JSON in the response
      res.json(responseData);
    }
    else {
      // handling the JSON response in the case the user hasn't uploaded any custom images
      const selectData = [];
      const extensions = [];
      const responseData = {
        selectData: selectData,
        imageExtensions: extensions
      };
      res.json(responseData);
    }
  })
});

```

Κώδικας 3.13 Το getSelectData route εντός του server.

```

// asynchronous function to update the <select> element with data from the server
function updateSelectWithData() {
  // make a request to the server route
  fetch('/getSelectData')
    .then((response) => response.json())
    .then((data) => {
      const selectElement = document.getElementById('selectmask');

      const selectData = data.selectData;
      const imageExtensions = data.imageExtensions;

      // add new options based on the data

      if (selectData !== undefined) {
        selectData.forEach((optionText, index) => {
          const extension = imageExtensions[index];

          // capitalize the first letter of optionText to match the format
          var capitalizedText = optionText.charAt(0).toUpperCase() + optionText.slice(1);

          const option = document.createElement('option');
          option.text = capitalizedText;

          // set the extension as a data attribute to each option
          option.setAttribute('data-extension', extension);

          // adding a value to each option starting from 17 as there are already 17 included images in the app
          option.value = option_value;
          option_value += 1;

          selectElement.appendChild(option);

        });
      }

      // after the custom images' names have been loaded to our selectElement
      // we create the container with the available options
      createOptionsContainer();
    })
    .catch((error) => {
      console.error('Error fetching data:', error);
    });
}

```

Κώδικας 3.14 Η ασύγχρονη συνάρτηση `updateSelectWithData()` .

3.8 Δεδομένα του `getImages` route

Η ενημέρωση του πίνακα `images` του `index.html` ο οποίος περιέχει το `id` (όνομα) και το `path` κάθε εικόνας γίνεται μέσω του `/getImages` route. Το `app.get('/getImages')` ανακτά (όπως και το `app.get('/getSelectData')`) τα `id` των εικόνων από τη βάση, ενώ παράλληλα παίρνει το πλήρες `path` τους καθώς και τον πίνακα με τις αποθηκευμένες συντεταγμένες τους από τη στήλη `coordinates` ([Κώδικας 3.15](#)). Τα 3 αυτά δεδομένα στέλνονται ως JSON response στο `client.js`, το οποίο κάνει `fetch` των δεδομένων μέσω της ασύγχρονης συνάρτησης `updateImages()` και αποθηκεύει το parsed JSON response στη μεταβλητή `data`. Από το `data` εξάγει το `id` και το `path` κάθε εικόνας και τα αποθηκεύει στον πίνακα `clientImages` τον οποίο

στη συνέχεια συνενώνει με τον πίνακα `images`. Για την εξαγωγή του πίνακα των συντεταγμένων η συνάρτηση χρησιμοποιεί τη μέθοδο `reduce` η οποία μετατρέπει τον πίνακα των συντεταγμένων κάθε εικόνας σε αντικείμενο στο οποίο κάθε `image id` είναι συσχετισμένο με τις συντεταγμένες που του αναλογούν (Κώδικας 3.16). Το αντικείμενο αυτό στέλνεται ως παράμετρος κατά την κλήση της συνάρτησης `handleClientImages()` (Κώδικας 3.17) η οποία βρίσκεται στο `index.html`. Η `handleClientImages()` ενημερώνει το αντικείμενο `masks` με τις επιπλέον συντεταγμένες των ανεβασμένων εικόνων και πραγματοποιεί τη φόρτωση της μάσκας κάθε εικόνας.

```
// define a route to fetch image data from the database (called in client.js)
app.get('/getImages', (req, res) => {
  const db = new sqlite3.Database('clm_database.db');
  userId = req.session.userId;
  const sql = 'SELECT filename, coordinates FROM user_images WHERE user_id = ?';
  db.all(sql, [req.session.userId], (err, rows) => {
    if (err) {
      console.error('Error fetching custom images:', err);
      res.status(500).send('Internal Server Error');
    }
    else if (rows && rows.length > 0) {
      // define the image directory
      const image_url = "../uploads/";

      // map rows to the desired format
      const imageData = rows.map((row) => ({
        id: path.parse(row.filename).name,
        path: image_url + row.filename,
        coordinates: row.coordinates // add the coordinates to the response
      }));
      // send the data as JSON in the response
      res.json(imageData);
    }
    else {
      // handling the JSON response in case the user hasn't uploaded any custom images
      const imageData = [];
      res.json(imageData);
    }
  });
});
```

Κώδικας 3.15 Το `getImages` route εντός του server.

```

// asynchronous function to update the images array in index.html
async function updateImages() {
  try {
    const response = await fetch('/getImages');
    const data = await response.json();

    const clientImages = data.map((item) => ({
      id: item.id,
      path: item.path
    }));

    const imageCoordinates = data.reduce((acc, item) => {
      // assuming that item.id contains the key (e.g., "audrey")
      const key = item.id;
      const coordinates = JSON.parse(item.coordinates);

      // assign the coordinates array to the key in the accumulator object
      acc[key] = coordinates;

      return acc;
    }, {});

    images = images.concat(clientImages);

    // once images are updated, call the function within index.html that relies on clientimages
    handleClientImages(imageCoordinates);
  } catch (error) {
    console.error('Error fetching image data:', error);
  }
}
updateImages();

```

Κώδικας 3.16 Η ασύγχρονη συνάρτηση updateImages().

```

// function to handle the updated images and masks the user has provided
function handleClientImages(imageCoordinates) {
    // update the masks object with the corresponding index of the coordinates array
    for (var name in imageCoordinates) {
        masks[name] = imageCoordinates[name];
    };
    console.log(images);
    console.log(imageCoordinates);
    console.log(masks);

    var imageCount = 0;
    loadMask = function(index) {
        var mask = new Image();
        mask.onload = function(obj) {
            var elementId = images[index].id;

            // copy the images to canvases
            imagecanvas = document.createElement('CANVAS');
            imagecanvas.width = obj.target.width;
            imagecanvas.height = obj.target.height;
            imagecanvas.getContext('2d', { willReadFrequently: true }).drawImage(obj.target, 0, 0);
            imagecanvas.imageSmoothingEnabled = true;
            imageCanvases[elementId] = imagecanvas;

            imageCount += 1;
            if (imageCount == images.length) {
                imagesReady = true;
            }
            enablestart();
        }
        mask.src = images[index].path;
    }
    //load masks
    for (var i = 0; i < images.length; i++) {
        loadMask(i);
    }
}

```

Κώδικας 3.17 Η συνάρτηση handleClientImages() η οποία λειτουργεί ως μεσάζων μεταξύ του client.js και του index.html.

3.9 Container με τις διαθέσιμες εικόνες

Όπως αναφέρθηκε, στο client.js γίνεται η αρχικοποίηση και προσθήκη όλων των λειτουργιών του container που εμφανίζει τα ονόματα των εικόνων. Στο τέλος της updateSelectData(), αφού έχει ενημερωθεί το selectmask, καλείται η συνάρτηση createOptionsContainer() ([Κώδικας 3.18](#)). Η συνάρτηση προσθέτει επαναληπτικά όλα τα στοιχεία του selectmask (δηλαδή τις ήδη υπάρχουσες και τις νέες εικόνες) στο options-container το οποίο εμφανίζεται στο χρήστη. Στα τελευταία options (δηλαδή τις ανεβασμένες εικόνες) ενσωματώνει ως attribute το πλήρες όνομα τους με το extension και προσθέτει δίπλα σε κάθε option ένα κουμπί για τη διαγραφή

τους. Η διαδικασία της διαγραφής πραγματοποιείται με τη μέθοδο `forEach()` η οποία ελέγχει για ποια εικόνα έχει πατηθεί το κουμπί και στέλνει το `attribute` με το πλήρες όνομά της στο `/discardUpload route` το οποίο διαγράφει την καταχώρηση της από το `database`.

Στην αρχική του μορφή, το `container` είναι κλειστό και εμφανίζει μόνο το όνομα της τρέχουσας επιλεγμένης εικόνας. Όταν ο χρήστης πατήσει πάνω στο `container` εκείνο ανοίγει και εμφανίζονται 4 ακόμα επιλογές. Εντός αυτού, ο χρήστης μπορεί να περιηγηθεί στις διαθέσιμες εικόνες κάνοντας `scroll` και στη συνέχεια κάνοντας κλικ πάνω στην επιλογή που επιθυμεί, ενώ έχει και τη δυνατότητα να αλλάξει σειριακά το πρόσωπο που εφαρμόζεται χρησιμοποιώντας τα βελάκια που βρίσκονται στην άνω και κάτω άκρη του `container`. Σε κάθε μια από τις δύο περιπτώσεις, με το που γίνει η επιλογή από το χρήστη καλείται ένα `dispatchEvent()` το οποίο ειδοποιεί το `selectmask` εντός του `index.html` ότι η εικόνα έχει αλλάξει και καλεί τις συναρτήσεις (`switchMasks`, `createMasking` κτλ) με τις οποίες πραγματοποιείται η διαδικασία εφαρμογής του νέου προσώπου στο πρόσωπο του χρήστη.

```

// function to create the optionsContainer based on the updated selectmask
function createOptionsContainer() {
    const optionsContainer = document.querySelector(".options-container");

    // get the index of the current image mask which is set randomly in index.html
    var currentMask = selectElement.selectedIndex;

    // iterate over the options in the select element
    for (let i = 0; i < selectElement.options.length; i++) {
        var option = selectElement.options[i];

        // create a new container element for each option
        var optionContainer = document.createElement('div');
        optionContainer.classList.add('option-container');

        // create a new span element for each option
        var newOption = document.createElement('span');

        newOption.classList.add('menu-item');
        newOption.textContent = option.textContent;

        var extension = option.getAttribute('data-extension');
        newOption.setAttribute('option-extension', extension);

        // add a "data-selected" attribute to the focused item
        if (i === currentMask) {
            newOption.setAttribute('data-selected', 'true');
        }

        // append the new span and deleteButton elements to the container
        optionContainer.appendChild(newOption);

        if (i >= 17) {
            // create a delete button for custom uploaded images
            var deleteButton = document.createElement('button');
            deleteButton.classList.add('delete-button');
            deleteButton.textContent = 'delete';

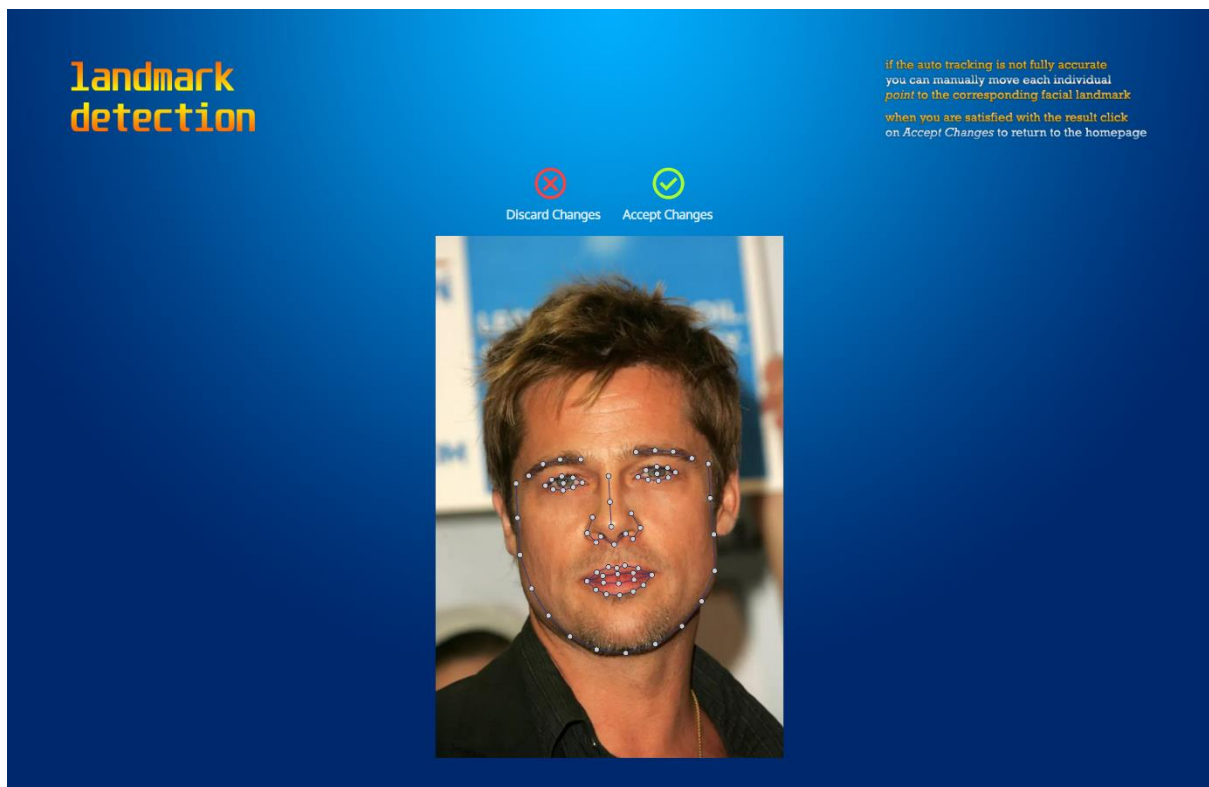
            deleteButton.style.fontFamily = '"Material Symbols Outlined", sans-serif';

            optionContainer.appendChild(deleteButton);
        }

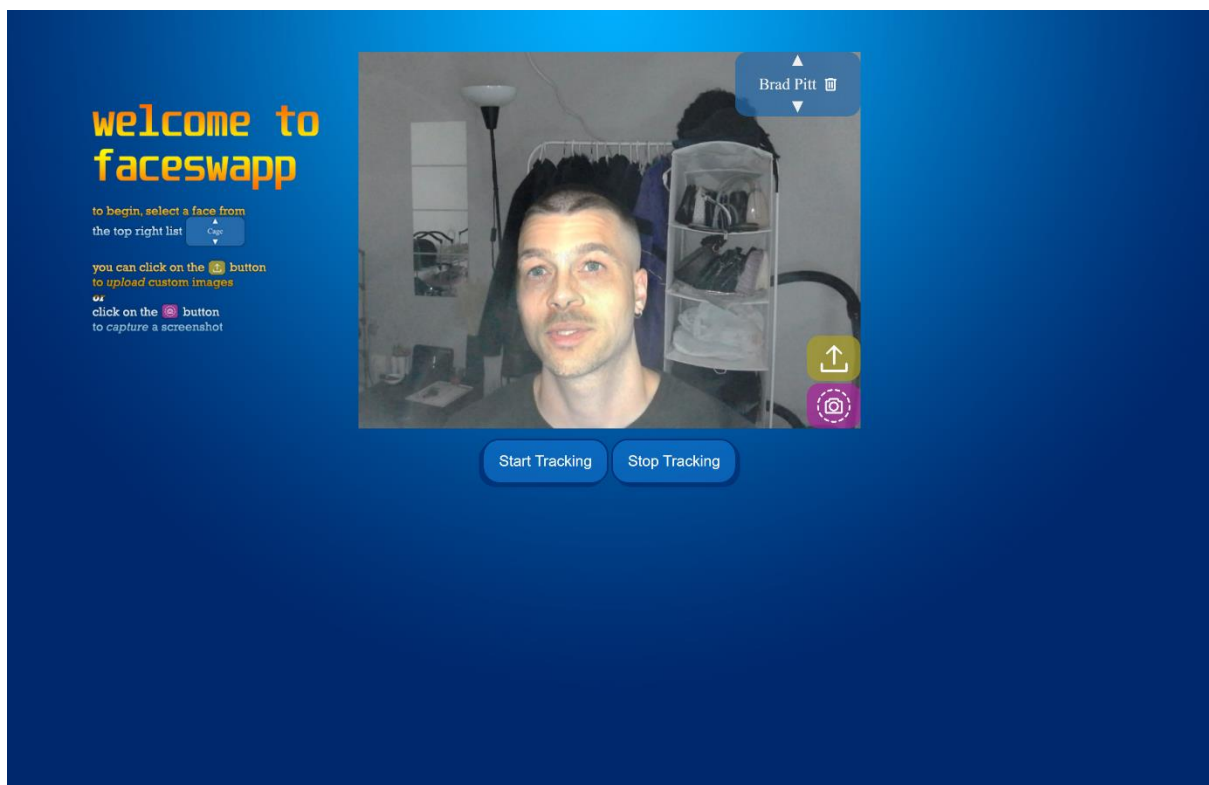
        // append the container to the optionsContainer
        optionsContainer.appendChild(optionContainer);
    }
}

```

Κώδικας 3.18 Μέρος της συνάρτησης createOptionsContainer() στο οποίο περιλαμβάνονται οι λειτουργίες που έχουν περιγραφεί.



Εικόνα 3.8 Το περιβάλλον του annotator με διαφορετική tracked εικόνα.



Εικόνα 3.9 Το εφαρμοσμένο πρόσωπο της Εικόνας 12 με το κεφάλι του χρήστη να έχει κλίση και μορφασμό.

3.10 Deployment/Εγκατάσταση

Η εφαρμογή είναι deployed live στο διαδίκτυο μέσω του Render¹¹ το οποίο κάνει την εφαρμογή host απευθείας μέσω των ανεβασμένων αρχείων στο GitHub¹². Για την εκτέλεση της εφαρμογής τοπικά το μόνο που απαιτείται είναι η εγκατάσταση του Node.js. Τρέχοντας την εντολή `node server.js` στον root φάκελο πραγματοποιείται εκτέλεση του server τοπικά με την εφαρμογή να είναι διαθέσιμη στη διεύθυνση `localhost:8080`.

¹¹ <https://clm-test.onrender.com>

¹² https://github.com/sudowicked/clm_test

4 Συμπεράσματα και Μελλοντικές Επεκτάσεις

Σκοπός της παρούσας εργασίας ήταν η δημιουργία μιας ολοκληρωμένης ψυχαγωγικής web εφαρμογής που δίνει στον χρήστη τη δυνατότητα της αλλαγής του προσώπου του σε πραγματικό χρόνο. Μέσα από τη μελέτη του κώδικα της βιβλιοθήκης Clmtrackr και του παραδείγματος face substitution που παρείχε ο δημιουργός της παρουσιάστηκαν οι πολλές και σύνθετες τεχνικές που απαιτούνται για να παραχθεί ένα όσο το δυνατό φυσικό για το ανθρώπινο μάτι αποτέλεσμα. Η υλοποίηση του backend της εφαρμογής επιτεύχθηκε με τη χρήση του Node.js στο οποίο με τα modules που παρέχει, όπως για τη δημιουργία του web server, οφείλεται το μεγαλύτερο μέρος της διαδικασίας του στησίματος της εφαρμογής στο διαδίκτυο. Για το περιβάλλον διεπαφής με το χρήστη έγινε χρήση APIs και συναρτήσεων της HTML και της JavaScript με σκοπό την προσομοίωση ενός φιλικού και εύληπτου περιβάλλοντος και την ομαλή εμπειρία των χρηστών κατά την πλοήγησή τους εντός της εφαρμογής, όπως και σε άλλες ψυχαγωγικές εφαρμογές. Η απλότητα και η ευελιξία που προσφέρει το Node.js διευκολύνει σημαντικά τη διαδικασία εγκατάστασης και εκτέλεσης του κώδικα της εφαρμογής, ενώ καθιστά τη βελτίωση και την προσθήκη επιπλέον λειτουργιών εύκολη για τον χρήστη.

Όσον αφορά το μοντέλο SPCA που χρησιμοποιείται για το fitting των προσώπων υπάρχουν βελτιώσεις που μπορούν να πραγματοποιηθούν στην εκπαίδευσή του. Εκτός από το database του MUCT η εκπαίδευσή του με περισσότερη ποικιλία annotated εικόνες προσώπων σε διαφορετικές πόζες και εκφράσεις αυξάνει τη πιθανότητα οι παραγόμενοι classifiers να εντοπίζουν ορθά το πρόσωπο σε εικόνες στις οποίες τα χαρακτηριστικά δεν είναι απόλυτα διακριτά. Παράλληλα, στην εφαρμογή μπορούν να προστεθούν επιπλέον περιβάλλοντα που θα παρέχουν και άλλες λειτουργίες της βιβλιοθήκης πέρα από την αλλαγή προσώπου, όπως το Realtime face deformation το οποίο επιτρέπει στο χρήστη να παραμορφώνει τα χαρακτηριστικά του μεταβάλλοντας τις παραμέτρους των χαρακτηριστικών του μοντέλου PCA που εφαρμόζεται στο πρόσωπό του. Έχοντας ως βάση τη βιβλιοθήκη Clmtrackr και αξιοποιώντας τις δυνατότητές της μπορούν να προστεθούν και άλλες παρόμοιες τεχνικές τροποποίησης βίντεο σε πραγματικό χρόνο προσομοιάζοντας έτσι ακόμα περισσότερο τις δημοφιλείς ψυχαγωγικές εφαρμογές των social media.

5 Βιβλιογραφία/Δικτυογραφία

- [1] “Instagram Wikipedia”, Διαθέσιμο στο δικτυακό τόπο: <https://en.m.wikipedia.org/wiki/Instagram>.
- [2] “Filters on Snapchat: What’s Behind The Augmented Reality Curtain”, Διαθέσιμο στο δικτυακό τόπο: <https://www.banuba.com/blog/snapchat-filter-technology-whats-behind-the-curtain>.
- [3] “TikTok Effect House”, Διαθέσιμο στο δικτυακό τόπο: <https://effecthouse.tiktok.com>.
- [4] “Meta Spark”, Διαθέσιμο στο δικτυακό τόπο: <https://spark.meta.com>.
- [5] “MDN Web APIs”, Διαθέσιμο στο δικτυακό τόπο: <https://developer.mozilla.org/en-US/docs/Web/API>.
- [6] “MDN Express Node”, Διαθέσιμο στο δικτυακό τόπο: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction
- [7] “Auduno.com: Clmtrackr Reference”, Διαθέσιμο στο δικτυακό τόπο: <https://www.auduno.com/clmtrackr/docs/reference.html>.
- [8] “Auduno.com: An Explanation of Clmtrackr”, Διαθέσιμο στο δικτυακό τόπο: <https://www.auduno.com/2014/01/05/fitting-faces/>.
- [9] “Auduno.com: Some Use Cases of Clmtrackr”, Διαθέσιμο στο δικτυακό τόπο: <https://www.auduno.com/2014/04/29/twisting-faces/>.
- [10] Jason M. Saragih, Simon Lucey, Jeffrey F. Cohn (2009). “Face Alignment through Subspace Constrained Mean-Shifts”, Διαθέσιμο στο δικτυακό τόπο: https://www.ri.cmu.edu/pub_files/2009/9/CameraReady-6.pdf.
- [11] “Node.js”, Διαθέσιμο στο δικτυακό τόπο: <https://nodejs.org/en>.
- [12] “SQLite”, Διαθέσιμο στο δικτυακό τόπο: <https://www.sqlite.org>.
- [13] “HTTP Request Methods”, Διαθέσιμο στο δικτυακό τόπο: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>.
- [14] “How To Deliver HTML Files with Express”, Διαθέσιμο στο δικτυακό τόπο: <https://www.digitalocean.com/community/tutorials/use-expressjs-to-deliver-html-files>.
- [15] “Pass Value from One Page to Another in JavaScript”, Διαθέσιμο στο δικτυακό τόπο: <https://xerosource.com/pass-value-from-one-page-to-another-in-javascript/>.
- [16] “MDN Async Function”, Διαθέσιμο στο δικτυακό τόπο: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function.
- [17] “How to Get a Picture Drawn In Canvas using Drawimage”, Διαθέσιμο στο δικτυακό τόπο: <https://stackoverflow.com/questions/46910780/how-to-get-a-picture-drawn-in-canvas-using-drawimage>.

- [18] “W3Schools HTML Forms”, Διαθέσιμο στο δικτυακό τόπο: https://www.w3schools.com/html/html_forms.asp.
- [19] David Cristinacce and Tim Cootes, “Feature Detection and Tracking with Constrained Local Models”, Διαθέσιμο στο δικτυακό τόπο: https://personalpages.manchester.ac.uk/staff/timothy.f.cootes/Papers/BMVC06/cristinacce_bmvc06.pdf.
- [20] “CSS Buttons”, Διαθέσιμο στο δικτυακό τόπο: <https://getcssscan.com/css-buttons-examples>.
- [21] “An Intuitive Explanation of Using Poisson Blending for Seamless Copy-and-Paste of Images”, Διαθέσιμο στο δικτυακό τόπο: https://erkaman.github.io/posts/poisson_blending.html.
- [22] “MDN Array Reduce”, Διαθέσιμο στο δικτυακό τόπο: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce.
- [23] “Downloading an image using XMLHttpRequest”, Διαθέσιμο στο δικτυακό τόπο: <https://stackoverflow.com/questions/8778863/downloading-an-image-using-xmlhttprequest-in-a-userscript>.
- [24] “SQLite Node.js”, Διαθέσιμο στο δικτυακό τόπο: <https://www.sqlitetutorial.net/sqlite-nodejs/>.
- [25] “Express Routing”, Διαθέσιμο στο δικτυακό τόπο: <https://expressjs.com/en/guide/routing.html>.
- [26] “How to Upload Images in Node.js using Multer”, Διαθέσιμο στο δικτυακό τόπο: <https://www.makeuseof.com/upload-image-in-nodejs-using-multer/>.
- [27] “App Icons”, Διαθέσιμο στο δικτυακό τόπο: <https://iconscout.com>.