

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή Εργασία

**Ανάπτυξη Εφαρμογών Web Υψηλής Απόδοσης
με τη Χρήση Τεχνολογιών WebAssembly**

Ντάνιελ Γιορντάνωβ Π2015105
Επιβλέπων: Μιχαήλ Στεφανιδάκης

Σεπτέμβριος 2022

Επιβλέπων

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής

Τριμελής Επιτροπή

Μιχαήλ Στεφανιδάκης,	Αναπληρωτής Καθηγητής
Θεόδωρος Ανδρόνικος,	Αναπληρωτής Καθηγητής
Φοίβος Μυλωνάς,	Αναπληρωτής Καθηγητής

Περίληψη

Η εργασία αυτή αναλύει την πρόσφατη εμφάνιση μιας νέας τεχνολογίας για τον ιστό που ονομάζεται WebAssembly, η οποία πρόκειται για έναν νέο τύπο κώδικα που μπορούν να εκτελέσουν οι φυλλομετρητές ιστού. Το πρώτο κεφάλαιο παρέχει ένα θεωρητικό πλαίσιο σχετικά με τα είδη και τις δυνατότητες των εφαρμογών ιστού από την εποχή της εμφάνισής τους μέχρι σήμερα, καθώς και κάποιες σημαντικές πληροφορίες για τη JavaScript. Το δεύτερο κεφάλαιο επικεντρώνεται σε μεγάλο βαθμό σε πληροφορίες που αφορούν την WebAssembly ως τεχνολογία, όπως ο λόγος της δημιουργίας της, πώς θα μπορούσε να χρησιμοποιηθεί, πώς μπορεί να γραφτεί και ποιες άλλες τεχνολογίες και γλώσσες την υποστηρίζουν, εντός ή εκτός από τους φυλλομετρητές ιστού. Το τρίτο κεφάλαιο υιοθετεί μια πιο πρακτική προσέγγιση αναπτύσσοντας μια διαδικτυακή εφαρμογή για την επεξεργασία εικόνων. Αναλύονται από πρώτο χέρι όλες οι πτυχές της διαδικασίας ανάπτυξης από την οπτική γωνία ενός αρχάριου προγραμματιστή JavaScript, ενώ παρέχονται επίσης πληροφορίες που αφορούν την εγκατάσταση της εφαρμογής, τη χρήση της, τις επιτυχίες και τα μειονεκτήματά της, καθώς και την πιθανή μελλοντική επεκτασιμότητα. Στο τελευταίο κεφάλαιο παρουσιάζονται τα συμπεράσματα της εργασίας σχετικά με την εν λόγω τεχνολογία. Συνολικά ήταν ως επί το πλείστον θετικά για την τρέχουσα κατάσταση της WebAssembly, λίγα χρόνια μετά την αρχική κυκλοφορίας της, γεγονός που οδηγεί σε πολύ ενδιαφέρουσες προοπτικές για το μέλλον. Η ανάπτυξη της WebAssembly αναμένεται να συνεχιστεί μέσα στα επόμενα χρόνια και υπάρχουν προσδοκίες για την ανάδυση νέων έργων που να αξιοποιούν αυτήν την τεχνολογία.

Abstract

This paper analyzes the recent emergence of a new web technology called WebAssembly, a new type of code that web browsers can execute. The first chapter provides some theoretical context on the types and capabilities of web applications from the time of their emergence until today, as well as important information about JavaScript. The second chapter heavily focuses on information surrounding WebAssembly as a technology, like its reason for creation, how it could be used, how it can be written and what other technologies and languages support it, inside or outside the browser. The third chapter takes on a more practical approach by developing a web application for image manipulation. It analyzes first-hand all aspects of the development process from the perspective of a novice JavaScript programmer, while also providing context pertaining to the app's installation, usage, successes and drawbacks, as well as future extensibility. The last chapter presents the final conclusions regarding to this technology. They were overall mostly positive about WebAssembly's current state, a few years after its introduction, a fact that leads to very exciting prospects for the future. WebAssembly's growth is expected to continue in the following years while more exciting and maturing projects are expected to emerge.

Πίνακας Περιεχομένων

1ο Κεφάλαιο:	Εισαγωγή	6
1.1:	Τα web applications ιστορικά	
1.2:	Ο ρόλος της JavaScript	
1.3:	Η WebAssembly	
2ο Κεφάλαιο:	Ανάπτυξη λογισμικού με WebAssembly	10
2.1:	Στόχοι και βασικές έννοιες	
2.2:	Υποστήριξη από browsers, benchmarks	
2.3:	Δομή και χρήση	
2.4:	Μορφές & Γλώσσες WebAssembly	
3ο Κεφάλαιο:	Μια πρακτική προσέγγιση: Επεξεργασία Εικόνας	26
3.1:	Τι είναι η επεξεργασία εικόνας και γιατί η WebAssembly ταιριάζει για αυτόν τον σκοπό	
3.2:	Εισαγωγή στην εφαρμογή	
3.3:	Εργαλεία και τεχνολογίες που χρησιμοποιήθηκαν	
3.4:	Η δομή των αρχείων	
3.5:	Εξήγηση της λογικής του κώδικα	
3.6:	Εμπόδια που αντιμετωπίστηκαν κατά την ανάπτυξη	
3.7:	Διαδικασία εγκατάστασης και χρήσης	
3.8:	Αποτελέσματα και δυνατότητες επέκτασης	
4ο Κεφάλαιο:	Συμπεράσματα και Μελλοντικές Βλέψεις	47
ΒΙΒΛΙΟΓΡΑΦΙΑ-ΔΙΚΤΥΟΦΡΑΦΙΑ.....		48

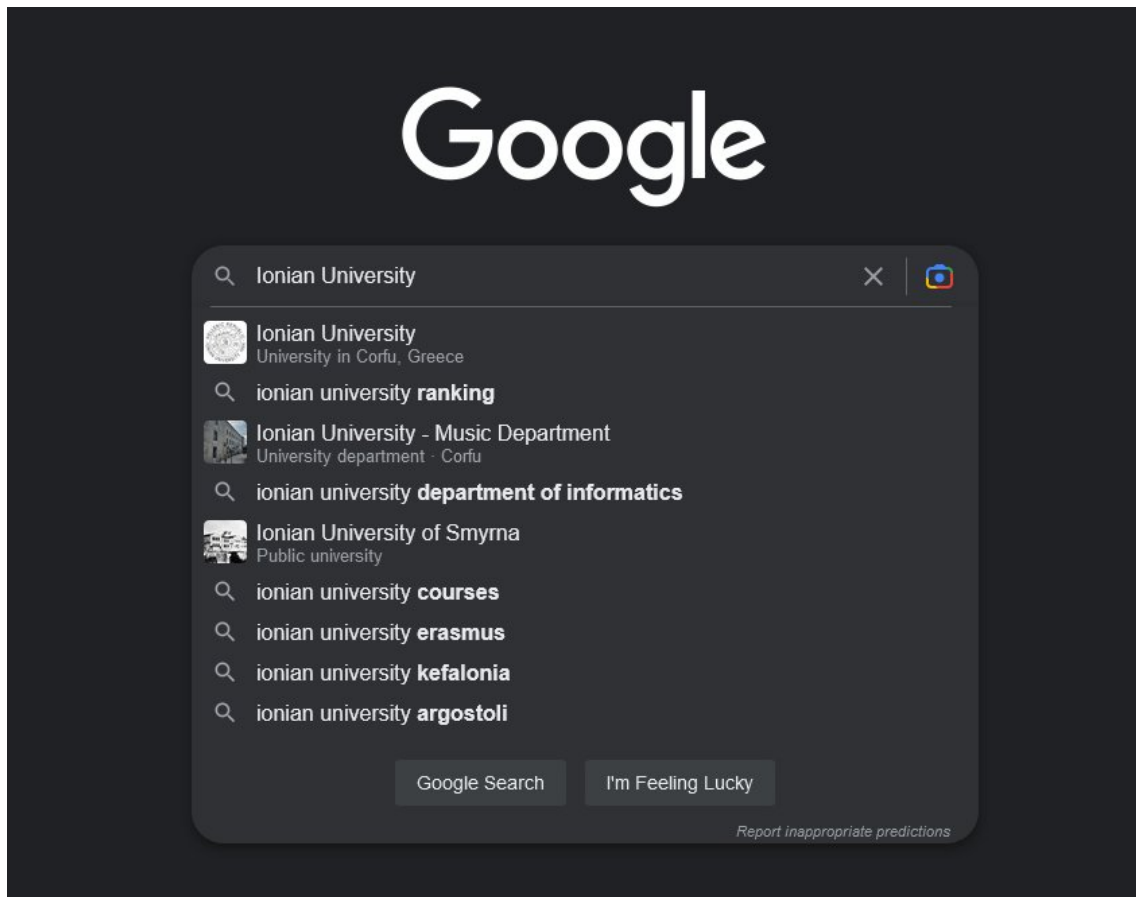
1ο Κεφάλαιο: Εισαγωγή

1.1: Τα web applications ιστορικά

Στο Διαδίκτυο των αρχών της δεκαετίας του '90, πριν τα προγράμματα περιήγησης ιστού (web browsers) αποκτήσουν τη δυνατότητα να εκτελούν προγραμματιστικά "σενάρια" (scripts), οι ιστοσελίδες της τότε εποχής ήταν απλά έγγραφα κειμένου, μερικές φορές με πρόσθετα αρχεία γραφικών, ήχου, εικόνων, ή βίντεο. Αυτές οι ιστοσελίδες ήταν αυτό που σήμερα αποκαλείται "στατικές ιστοσελίδες" (static websites), δηλαδή ιστοσελίδες που παραδίδονται στον web browser του χρήστη ακριβώς όπως είναι αποθηκευμένες στον εξυπηρετητή (server) από τον οποίο λήφθηκαν. Εφαρμογές υπολογιστών όπως προγράμματα email, επεξεργαστές κειμένου, μετατροπείς αρχείων, κλπ., υπήρχαν μόνο ως "εφαρμογές επιφάνειας εργασίας" (desktop applications), δηλαδή λογισμικό που είναι εγκατεστημένο στον σκληρό δίσκο του υπολογιστή του χρήστη.

Αυτό άρχισε να αλλάζει με αργούς ρυθμούς το 1995 όταν η εταιρεία Netscape εισήγαγε μια γλώσσα προγραμματισμού από την "πλευρά του πελάτη" (client-side) που ονομάζεται JavaScript. Αυτή επέτρεψε στους προγραμματιστές να προσθέσουν ορισμένα δυναμικά στοιχεία στη διεπαφή χρήστη που εκτελούνταν από την "πλευρά του πελάτη", δηλαδή τον web browser. Έτσι, αντί να αποστέλλονται δεδομένα στον server για να δημιουργηθεί μια ολόκληρη ιστοσελίδα, τα ενσωματωμένα scripts της ληφθείσας ιστοσελίδας μπορούν να εκτελέσουν διάφορες εργασίες, όπως επικύρωση εισόδου (input validation) ή εμφάνιση/απόκρυψη τμημάτων της σελίδας. Μια συνέχεια αυτής της αλλαγής έγινε στις αρχές του 2000 όταν το παρουσιάστηκε και τυποποιήθηκε η Ajax (Asynchronous JavaScript), ένα σύνολο από τεχνικές ανάπτυξης ιστού (web development) που αποσκοπούν για τη δημιουργία ασύγχρονων εφαρμογών ιστού (asynchronous web applications). Με τον διαχωρισμό του επιπέδου ανταλλαγής δεδομένων (data interchange layer) από το επίπεδο παρουσίασης (presentation layer), η Ajax επιτρέπει στις ιστοσελίδες και, κατ' επέκταση, στις εφαρμογές ιστού (web applications/apps), να αλλάζουν δυναμικά το περιεχόμενο χωρίς την ανάγκη επαναφόρτωσης ολόκληρης της σελίδας. Αυτό οδήγησε τελικά στη δημιουργία πολλών

web apps (π.χ. Gmail, Google Maps) που άρχισαν να μοιάζουν με παραδοσιακά desktop applications.



Στιγμιότυπο 1: Ένα παράδειγμα Ajax είναι η δυνατότητα αυτόματης συμπλήρωσης στην αναζήτηση Google. Αυτή η λειτουργία προσφέρει προτάσεις αναζήτησης, βοηθώντας τους χρήστες να ολοκληρώσουν το ερώτημα τους όταν πληκτρολογούν στη γραμμή αναζήτησης. Οι αλλαγές γίνονται σε πραγματικό χρόνο, αλλά η ιστοσελίδα παραμένει η ίδια.

Οι ιστοσελίδες δεν είναι πια απλά έγγραφα εδώ και πολύ καιρό. Στο διαδίκτυο υπάρχει πλέον μια τεράστια ποικιλία από πολύ δημοφιλή web applications, όπως προγράμματα ηλεκτρονικού ταχυδρομείου (web clients), σουίτες γραφείου (office suites), διαδραστικούς χάρτες, κοινωνικών δικτύων (social media), ηλεκτρονικά καταστήματα (e-shops), υπηρεσίες "ροής"(streaming) ταινιών και μουσικής, εφαρμογές διοίκησης και διαχείρισης έργων (project management) και πολλά ακόμη. Οι εφαρμογές web συνεχίζουν να βελτιώνονται με την πάροδο του χρόνου, αποκτώντας όλο και περισσότερες δυνατότητες και βελτιωμένη εμπειρία χρήστη, και συχνά προτιμώνται ακόμη και από εδραιωμένα desktop applications για τα πλεονεκτήματά τους.

1.2: Ο ρόλος της JavaScript

Σήμερα η JavaScript υποστηρίζεται από όλους τους διαδεδομένους browsers ως η κυρίαρχη client-side γλώσσα προγραμματισμού. Αυτό την καθιστά (μαζί με την HTML και την CSS) μια από τις βασικές τεχνολογίες του Παγκόσμιου Ιστού (World Wide Web, WWW) και σχεδόν κάθε ιστότοπος στο Διαδίκτυο (Internet) προσφέρει κάποιου είδους λειτουργικότητας JavaScript. Ωστόσο, το τοπίο του διαδικτύου έχει αλλάξει πάρα πολύ από τα χρόνια της αρχικής υιοθέτησης και ανάπτυξής του. Υπάρχουν πλέον σύγχρονες ανάγκες και προκλήσεις για web applications, όπως η αναγνώριση εικόνας, η εικονική/επαυξημένη πραγματικότητα, η επεξεργασία εικόνας/βίντεο, οι τρισδιάστατες απεικονίσεις, ο σχεδιασμός με τη βοήθεια υπολογιστή (CAD), τα βιντεοπαιχνίδια, η ισχυρή κρυπτογράφηση, κλπ., λειτουργίες για τις οποίες δεν σχεδιάστηκε η JavaScript. Το κοινό χαρακτηριστικό των παραπάνω παραδειγμάτων είναι ότι αποτελούν πολύπλοκες λειτουργίες που απαιτούν ιδιαίτερη επεξεργαστική ισχύ από τη φύση τους, όπως και γρήγορη και προβλέψιμη απόδοση από τον browser.

Η JavaScript είναι μια γλώσσα που διερμηνεύεται δυναμικά (dynamically interpreted language). Αυτό σημαίνει ότι σε αντίθεση με τις στατικά διερμηνευόμενες γλώσσες (statically interpreted language), ο διερμηνέας (interpreter) πρέπει να "μαντέψει" τον τύπο των μεταβλητών που χρησιμοποιούνται στον κώδικα και αυτό προσθέτει επιβάρυνση στην απόδοση. Και σε αντίθεση με τις μεταγλωττισμένες γλώσσες (compiled languages) που παράγουν μια μηχανικά αναγνώσιμη δυαδική μορφή, ο διερμηνέας πρέπει να αναλύσει, να μεταγλωττίσει και να βελτιστοποιήσει τον κώδικα ακριβώς πριν εκτελεστεί στον client. Αυτή η προσέγγιση έχει τα πλεονεκτήματά της, αλλά αυτά επιτυγχάνονται με κόστος στην ταχύτητα και στον χρόνο φόρτωσης του προγράμματος. Παρόλο που έχουν γίνει σημαντικές βελτιώσεις για την αντιμετώπιση αυτών των προβλημάτων (η JavaScript είναι πλέον μια ώριμη γλώσσα) όπως η "JavaScript μεταγλωττισμένη τη στιγμή της εκτέλεσης" (Just-in-Time, JIT compiled JavaScript), δεν θα έχει ποτέ την χρονικά προβλέψιμη λειτουργία μιας μεταγλωττισμένης γλώσσας. Αυτός είναι ένας από τους μεγαλύτερους λόγους για τους οποίους δημιουργήθηκε μια νέα τεχνολογία, που ονομάζεται WebAssembly.

1.3: Η WebAssembly

Η WebAssembly (WASM, WA) είναι μια δυαδική μορφή εντολών (binary instruction format), σχεδιασμένη ως φορητός στόχος μεταγλώττισης (portable compilation target) για γλώσσες προγραμματισμού. Ο κύριος στόχος της είναι να τρέχει εφαρμογές υψηλής απόδοσης σε ιστοσελίδες, αλλά ταυτόχρονα δεν απαιτεί εξειδικευμένα χαρακτηριστικά του ιστού, ώστε να μπορεί να χρησιμοποιηθεί και σε άλλα περιβάλλοντα. Ανακοινώθηκε για πρώτη φορά το 2015, μια πρώτη έκδοση κυκλοφόρησε το 2017 και στις 5 Δεκεμβρίου 2019 παρουσιάστηκε ως επίσημο πρότυπο ιστού (web standard) από την World Wide

Web Consortium (W3C). Σήμερα υποστηρίζεται από όλους τους διαδεδομένους browsers.

Μέσω αυτής της νέας τεχνολογίας, ένας προγραμματιστής μπορεί τώρα να γράψει κώδικα χαμηλού επιπέδου (π.χ. σε γλώσσα C), τον οποίο στη συνέχεια μπορεί να μεταγλωττίσει σε μορφή WASM, αναγνώσιμη και εκτελέσιμη από τον client. Αυτό φυσικά σημαίνει επίσης ότι οι υπάρχουσες βάσεις κώδικα μπορούν να μεταγλωττιστούν σε WASM και να αναπτυχθούν στον Ιστό. Μέσω της μεταγλώττισης παράγεται ένα δυαδικό αρχείο μικρού μεγέθους που εξοικονομεί bandwidth και οδηγεί σε μεγαλύτερες ταχύτητες μεταφοράς. Έχει ασφαλή χαρακτηριστικά χρήσης της μνήμης (memory-safe) και εκτελείται σε sandbox. Και το πιο σημαντικό, διαθέτει προβλέψιμη απόδοση, συγκρίσιμη με αυτή ενός desktop application. Σε αρκετές περιπτώσεις έχει παρατηρηθεί κατά μέσο όρο αύξηση απόδοσης 30%¹ σε σχέση με τη JavaScript, με αποκλίσεις ανάλογα με τον browser που χρησιμοποιείται, τον τύπο της εφαρμογής που έχει αναπτυχθεί και τη γλώσσα που χρησιμοποιείται για τη μεταγλώττιση.

Η συγκεκριμένη εργασία έχει ως στόχο να εντοπίσει τα πλεονεκτήματα της WebAssembly, να συγκρίνει την απόδοσή της με τη JavaScript σε διαφορετικά σενάρια, και να αναλύσει περιπτώσεις χρήσης της για την ανάπτυξη αποτελεσματικών και γρήγορων web applications.

¹ Ένα τέτοιο παράδειγμα είναι το [“WasmBoy”](#), ένας εξομοιωτής WebAssembly ενός GameBoy Color.

2ο Κεφάλαιο: Ανάπτυξη λογισμικού με WebAssembly

Η WebAssembly είναι ένας νέος τύπος κώδικα που μπορεί να εκτελεστεί στους σύγχρονους φυλλομετρητές ιστού - είναι μια γλώσσα χαμηλού επιπέδου που μοιάζει με assembly με συμπαγή δυαδική μορφή που εκτελείται με σχεδόν εγγενείς (native) επιδόσεις και παρέχει σε γλώσσες όπως την C/C++, την C# και την Rust έναν στόχο μεταγλώττισης (compilation target) ώστε να μπορούν να εκτελούνται στην πλατφόρμα του διαδικτύου. Έχει επίσης σχεδιαστεί για να τρέχει παράλληλα με τη JavaScript, επιτρέποντας τη συνεργασία και των δύο. Αυτό έχει τεράστιες επιπτώσεις για την πλατφόρμα του διαδικτύου επειδή παρέχει έναν τρόπο εκτέλεσης κώδικα γραμμένο σε πολλαπλές γλώσσες στον ιστό με σχεδόν εγγενή ταχύτητα, με εφαρμογές ιστού να τρέχουν στον client/browser που προηγουμένως δεν μπορούσαν να το καταφέρουν.

Επιπλέον, ο προγραμματιστής δεν χρειάζεται καν να γνωρίζει πώς να γράφει κώδικα WebAssembly για να εκμεταλλευτεί την τεχνολογία. Δεν προορίζεται πρωτίστως για να γραφτεί "με το χέρι", αλλά έχει σχεδιαστεί για να αποτελεί έναν αποτελεσματικό στόχο μεταγλώττισης για άλλες γλώσσες προγραμματισμού. Οι ενότητες (modules) WebAssembly μπορούν να εισαχθούν σε μια εφαρμογή web (ή Node.js), εκθέτοντας συναρτήσεις (functions) WebAssembly για χρήση μέσω JavaScript. Τα υπάρχοντα frameworks της JavaScript θα μπορούσαν να κάνουν χρήση της WebAssembly για να προσδώσουν τεράστια πλεονεκτήματα απόδοσης και νέα χαρακτηριστικά, ενώ παράλληλα θα έκαναν τη λειτουργικότητά της εύκολα διαθέσιμη στους προγραμματιστές του ιστού.

Τέλος, σημαντικό είναι να επισημανθεί ότι αναπτύσσεται ως πρότυπο ιστού (web standard) μέσω των W3C WebAssembly Working Group και Community Group, με ενεργή συμμετοχή από όλους τους μεγαλύτερους παρόχους προγραμμάτων περιήγησης ιστού.

2.1: Στόχοι και βασικές έννοιες

Η WebAssembly αναπτύσσεται ως ανοικτό πρότυπο στο πλαίσιο της W3C WebAssembly Community Group με τους ακόλουθους στόχους:

1. "Be fast, efficient, and portable" - Ο κώδικας WebAssembly μπορεί να εκτελείται με σχεδόν εγγενή ταχύτητα σε διαφορετικές πλατφόρμες, εκμεταλλευόμενος κοινές δυνατότητες του υλικού.
2. "Be readable and debuggable" - Η WebAssembly είναι μια γλώσσα πολύ χαμηλού επιπέδου, αλλά διαθέτει μια μορφή κειμένου αναγνώσιμη από τον άνθρωπο (η προδιαγραφή της οποίας βρίσκεται ακόμη υπό οριστικοποίηση) που επιτρέπει τη συγγραφή, την προβολή και την αποσφαλμάτωση του κώδικα άμεσα πάνω στον κώδικα.
3. "Keep secure" - Η WebAssembly εκτελείται σε ένα ασφαλές, sandboxed περιβάλλον. Όπως και για τον υπόλοιπο κώδικα του ιστού, επιβάλλονται οι πολιτικές ασφαλείας του browser (δηλαδή same-origin και permissions).
4. "Don't break the web" - Η WebAssembly έχει σχεδιαστεί έτσι ώστε να μπορεί να συνεργάζεται με άλλες τεχνολογίες ιστού και να διατηρεί συμβατότητα με παλαιότερες εκδόσεις (backwards compatibility).

Για να εξηγηθεί πώς η Wasm ταιριάζει στον ιστό, πρέπει πρώτα να γίνει κατανοητή η πλατφόρμα. Η πλατφόρμα του ιστού μπορεί να θεωρηθεί ότι έχει δύο μέρη: Μια εικονική μηχανή (Virtual Machine, VM) που εκτελεί τον κώδικα της εφαρμογής ιστού (π.χ. τον κώδικα JavaScript που παρέχει την λειτουργικότητα των εφαρμογών), και ένα σύνολο από διεπαφές προγραμματισμού εφαρμογών (application programming interface, API) του ιστού που η εφαρμογή ιστού μπορεί να καλέσει για να ελέγξει τη συμπεριφορά του browser ή της συσκευής και να προσφέρει λειτουργικότητα (DOM, CSSOM, WebGL, IndexedDB, Web Audio API, κ.λπ.).

Ιστορικά, το VM ήταν σε θέση να φορτώνει μόνο JavaScript. Αυτό συμφέρει τους προγραμματιστές ιστού, καθώς δεν χρειάζεται να γνωρίζουν πολλαπλές γλώσσες προγραμματισμού και η JavaScript είναι αρκετά ισχυρή για να καλύψει τις ανάγκες των ανθρώπων στον ιστό σήμερα. Ωστόσο, παρουσιάζονται προβλήματα επιδόσεων όταν γίνονται προσπάθειες χρήσης της JavaScript για πιο απαιτητικές περιπτώσεις χρήσης, όπως τρισδιάστατα παιχνίδια, εικονική και επαυξημένη πραγματικότητα, όραση υπολογιστών, επεξεργασία εικόνας ή βίντεο και μια σειρά από άλλους τομείς που απαιτούν υψηλές επιδόσεις. Επιπλέον σε τέτοιες περιπτώσεις, το κόστος λήψης, ανάλυσης και μεταγλώττισης πολύ μεγάλων εφαρμογών JavaScript μπορεί να είναι απαγορευτικό. Τα smartphones και άλλες κινητές πλατφόρμες με περιορισμένους πόρους μπορούν να αυστηροποιήσουν περαιτέρω αυτές τις ανάγκες επιδόσεων.

Η WebAssembly είναι μια διαφορετική γλώσσα από τη JavaScript, αλλά δεν προορίζεται ως αντικαταστάτης. Αντίθετα, έχει σχεδιαστεί για να συμπληρώνει και να λειτουργεί παράλληλα με τη JavaScript, επιτρέποντας στους προγραμματιστές ιστού να επωφεληθούν από τα πλεονεκτήματα και των δύο γλωσσών:

- Η JavaScript είναι μια γλώσσα υψηλού επιπέδου, αρκετά ευέλικτη και εκφραστική για τη συγγραφή εφαρμογών ιστού. Έχει πολλά πλεονεκτήματα - είναι dynamically typed, δεν απαιτεί μεταγλώττιση και διαθέτει ένα τεράστιο οικοσύστημα που παρέχει ισχυρά frameworks, libraries και άλλα εργαλεία.
- Η WebAssembly είναι μια γλώσσα χαμηλού επιπέδου που μοιάζει με assembly με συμπαγή δυαδική μορφή που εκτελείται με σχεδόν native απόδοση και παρέχει σε γλώσσες με low-level memory models, όπως η C++ και η Rust, έναν στόχο μεταγλώττισης, ώστε να μπορούν να τρέξουν στον ιστό. Στο μέλλον η WebAssembly έχει ως στόχο να υποστηρίξει γλώσσες με garbage-collected memory models.

Με την εμφάνιση της WebAssembly στους browsers, το VM που αναφέρθηκε νωρίτερα θα φορτώνει και θα εκτελεί πλέον δύο τύπους κώδικα - JavaScript και WebAssembly.

Οι διαφορετικοί τύποι κώδικα μπορούν να καλούν ο ένας τον άλλον όπως απαιτείται - το JavaScript API της WebAssembly τυλίγει τον εξαγόμενο κώδικα WebAssembly με συναρτήσεις JavaScript που μπορούν να κληθούν, και ο κώδικας WebAssembly μπορεί να εισάγει και να καλεί συγχρονισμένα κανονικές συναρτήσεις JavaScript. Στην πραγματικότητα, η βασική μονάδα του κώδικα WebAssembly ονομάζεται module και τα modules WebAssembly είναι συμμετρικά από πολλές απόψεις με τα ECMAScript (ES) modules.

Υπάρχουν αρκετές βασικές έννοιες που απαιτούνται για να κατανοηθεί πώς εκτελείται η WebAssembly στον browser και όλες τους αντικατοπτρίζονται μία προς μία στο WebAssembly JavaScript API. Τέσσερις από αυτές δίνονται παρακάτω:

- **Module:** Αντιπροσωπεύει ένα δυαδικό αρχείο WebAssembly που έχει μεταγλωττιστεί από τον browser σε εκτελέσιμο κώδικα μηχανής. Ένα Module είναι stateless και επομένως (παρόμοια με ένα Blob του ES) μπορεί να διαμοιραστεί ρητά μεταξύ παραθύρων και εργαζομένων (workers) μέσω `postMessage()`. Ένα Module δηλώνει εισαγωγές και εξαγωγές όπως ακριβώς ένα module του ES.

- **Memory:** Ένα ArrayBuffer μεταβλητού ή αμετάβλητου μεγέθους που περιέχει τον γραμμικό πίνακα (linear array) των bytes που διαβάζονται και γράφονται από τις εντολές πρόσβασης στη μνήμη χαμηλού επιπέδου της WebAssembly.
- **Table:** Ένας τυποποιημένος πίνακας αναφορών (π.χ. σε συναρτήσεις) μεταβλητού ή αμετάβλητου μεγέθους που δεν θα μπορούσαν διαφορετικά να αποθηκευτούν ως ακατέργαστα bytes στο Memory (για λόγους ασφάλειας και φορητότητας).
- **Instance:** Ένα module που συνδυάζεται με όλη την κατάσταση (state) που χρησιμοποιεί κατά την εκτέλεση, συμπεριλαμβανομένων ενός Memory, ενός Table και ενός συνόλου εισαγόμενων τιμών/μεταβλητών. Ένα Instance είναι σαν ένα module ES που έχει φορτωθεί σε ένα συγκεκριμένο WebAssembly Global object με ένα συγκεκριμένο σύνολο εισαγωγών.

Το JavaScript API παρέχει στους προγραμματιστές τη δυνατότητα να δημιουργούν modules, memories, tables και instances. Σε ένα instance WebAssembly, ο κώδικας JavaScript μπορεί να καλεί συγχρονισμένα τις εξαγωγές από αυτό το instance, οι οποίες εκτίθενται ως κανονικές συναρτήσεις JavaScript. Αυθαίρετες συναρτήσεις JavaScript μπορούν επίσης να κληθούν συγχρονισμένα από τον κώδικα WebAssembly περνώντας αυτές τις συναρτήσεις JavaScript ως εισαγωγές σε ένα instance WebAssembly.

Δεδομένου ότι η JavaScript έχει πλήρη έλεγχο του τρόπου λήψης, μεταγλώττισης και εκτέλεσης του κώδικα WebAssembly, οι προγραμματιστές JavaScript θα μπορούσαν να θεωρήσουν την WebAssembly ακόμη και ως ένα απλό χαρακτηριστικό της JavaScript για την αποτελεσματική παραγωγή συναρτήσεων/κώδικα υψηλής απόδοσης.

Στο μέλλον, τα modules WebAssembly θα μπορούν να φορτωθούν όπως ακριβώς και τα modules ES (χρησιμοποιώντας `<script type='module'>`), πράγμα που σημαίνει ότι η JavaScript θα μπορεί να αντλεί, να μεταγλωττίζει και να εισάγει ένα module WebAssembly τόσο εύκολα όσο και ένα module ES.

2.2: Υποστήριξη από browsers, benchmarks

Ο Mozilla Firefox και ο Google Chrome ήταν οι πρώτοι browsers που υποστήριζαν την WebAssembly, αρχικά μέσω της ενεργοποίησης πειραματικών σημαιών (experimental flags) στις προηγμένες ρυθμίσεις τους. Αργότερα υποστηρίχθηκαν επίσημα με την έκδοση 52 του Firefox, που κυκλοφόρησε στις 7 Μαρτίου 2017, και την έκδοση 57 του Chrome, που κυκλοφόρησε δύο ημέρες αργότερα. Στα τέλη του ίδιου έτους, η Apple

πρόσθεσε επίσης υποστήριξη WebAssembly στον browser της Safari με έκδοση 11 και το ίδιο έκανε και η Microsoft με την έκδοση 16 του Edge. Αυτό οδήγησε λίγο καιρό αργότερα στην επίσημη ανακοίνωση της Mozilla ότι η WebAssembly υποστηρίζεται πλέον επίσημα σε όλους τους μεγάλους browsers (για υπολογιστές). Σήμερα αυτό περιλαμβάνει και όλους τους μεγάλους browsers για συσκευές όπως τα smartphones και τα tablets.

Αυτό σημαίνει ότι ένας προγραμματιστής μπορεί πλέον να στοχεύει τον ιστό ως πλατφόρμα χρησιμοποιώντας το WebAssembly, όχι μόνο σε μια συγκεκριμένη συσκευή ή browser, και η JavaScript δεν είναι πλέον η μόνη επιλογή. Ένα τέτοιο παράδειγμα είναι ένα έργο ανοιχτού κώδικα που ονομάζεται WasmBoy, ένας εξομοιωτής (emulator) Game Boy Color γραμμένος για WebAssembly χρησιμοποιώντας AssemblyScript. Αν και το έργο βρίσκεται ακόμη σε στάδιο Alpha, ο δημιουργός του έχει δημοσιεύσει συγκριτικά στοιχεία του έργου του από το 2018 που δείχνουν σημαντική αύξηση των επιδόσεων σε σύγκριση με τη JavaScript σε όλες τις περιπτώσεις, και ειδικά σε κινητές συσκευές.

2.3: Δομή και χρήση

Το οικοσύστημα της WebAssembly βρίσκεται σε ακόμη σε αρχικό στάδιο και αναμφίβολα θα εμφανιστούν περισσότερα εργαλεία στο μέλλον. Αυτή τη στιγμή, υπάρχουν τέσσερα κύρια σημεία εισόδου.

- Η μεταφορά μιας εφαρμογής C/C++ με το Emscripten.
- Η συγγραφή μιας εφαρμογής Rust με στόχο μεταγλώττισης την WebAssembly.
- Η χρήση της AssemblyScript, μια γλώσσα που μοιάζει με την TypeScript και μεταγλωττίζει σε WebAssembly δυαδικής μορφής.
- Η απευθείας συγγραφή σε WebAssembly μορφής κειμένου.

Δύο από τις πολλές επιλογές για τη δημιουργία κώδικα WASM από C/C++ είναι ένας online Wasm assembler ή το Emscripten. Υπάρχουν πολλές επιλογές online Wasm assembler, όπως το WasmFiddle, το WasmFiddle++ και το WasmExplorer. Αυτά είναι εξαιρετικοί πόροι για όσους προσπαθούν να καταλάβουν από πού να ξεκινήσουν, αλλά θα τους λείπουν ορισμένα από τα εργαλεία και τις βελτιστοποιήσεις του Emscripten. Το εργαλείο Emscripten είναι σε θέση να λάβει σχεδόν οποιονδήποτε πηγαίο κώδικα C/C++ και να τον μεταγλωττίσει σε module Wasm (αρχείο με επέκταση .wasm), καθώς και τον απαραίτητο κώδικα JavaScript για τη φόρτωση και την εκτέλεση του module και ένα έγγραφο HTML για την εμφάνιση των αποτελεσμάτων του κώδικα.

Με λίγα λόγια, η διαδικασία λειτουργεί ως εξής:

1. Το Emscripten τροφοδοτεί πρώτα την C/C++ στο Clang/LLVM, μια σειρά από εργαλεία μεταγλώττισης C/C++ ανοικτού κώδικα, που παρέχεται ως μέρος του XCode στο OSX για παράδειγμα.
2. Το Emscripten μετατρέπει το αποτέλεσμα της μεταγλώττισης του Clang/LLVM σε ένα δυαδικό αρχείο με επέκταση .wasm.
3. Από μόνη της, η WebAssembly δεν μπορεί επί του παρόντος να έχει άμεση πρόσβαση στο DOM (Document Object Model). Μπορεί μόνο να καλέσει τη JavaScript, περνώντας πρωτογενείς τύπους δεδομένων, ακέραιους ή κινητής υποδιαστολής. Έτσι, για να αποκτήσει πρόσβαση σε οποιοδήποτε API του Ιστού, η WebAssembly πρέπει να καλέσει τη JavaScript, η οποία στη συνέχεια πραγματοποιεί την κλήση του API του Ιστού. Έπειτα, το Emscripten δημιουργεί τον "συγκολλητικό" κώδικα HTML και JavaScript που απαιτείται για να επιτευχθεί αυτό.

Αυτός ο "συγκολλητικός" κώδικας JavaScript δεν είναι τόσο απλός όσο ακούγεται. Για αρχή, το Emscripten υλοποιεί δημοφιλείς βιβλιοθήκες (libraries) C/C++ όπως τα SDL, OpenGL, OpenAL καθώς και τμήματα του POSIX. Αυτά τα libraries υλοποιούνται με όρους Web APIs και έτσι το κάθε ένα από αυτά απαιτεί κάποιο "συγκολλητικό" κώδικα JavaScript για τη σύνδεση της WebAssembly με το υποκείμενο Web API. Έτσι, μέρος του κώδικα υλοποιεί τη λειτουργικότητα κάθε αντίστοιχης βιβλιοθήκης που χρησιμοποιείται από τον κώδικα C/C++. Ο κώδικας JavaScript περιέχει επίσης τη λογική για την κλήση των προαναφερθέντων WebAssembly JavaScript APIs για την ανάκτηση, τη φόρτωση και την εκτέλεση του αρχείου .wasm.

Το παραγόμενο έγγραφο HTML φορτώνει το αρχείο JavaScript και γράφει την έξοδο stdout σε ένα `<textarea>`. Εάν η εφαρμογή χρησιμοποιεί OpenGL, η HTML περιέχει επίσης ένα element `<canvas>` που χρησιμοποιείται ως στόχος rendering. Από αυτό το θεμέλιο είναι εύκολο να τροποποιηθεί η έξοδος του Emscripten και να μετατραπεί σε όποια εφαρμογή ιστού επιθυμεί να υλοποιήσει ο προγραμματιστής.

Είναι επίσης δυνατό να χρησιμοποιηθεί κώδικας Rust για μεταγλώττιση σε WebAssembly, χάρη στην δουλειά της Rust WebAssembly Working Group. Ένα παράδειγμα χρήσης ξεκινάει με την εγκατάσταση των απαραίτητων εργαλείων, ακολουθούμενη από την μεταγλώττιση ενός προγράμματος από Rust σε ένα WebAssembly npm package, και τελικά την χρήση αυτού σε ένα web app. Η διαδικασία περιγράφεται πιο λεπτομερώς σε επόμενο κεφάλαιο.

Για τους προγραμματιστές web που θέλουν να δοκιμάσουν την WebAssembly χωρίς να χρειάζεται να μάθουν τις λεπτομέρειες της C ή της Rust, παραμένοντας στην άνεση μιας οικείας γλώσσας όπως η TypeScript, η AssemblyScript είναι μια καλή επιλογή. Η AssemblyScript μεταγλωττίζει ένα ελαφρώς παραλλαγμένο υποσύνολο της TypeScript σε WebAssembly, επιτρέποντας στους προγραμματιστές web να συνεχίσουν να χρησιμοποιούν εργαλεία συμβατά με TypeScript με τα οποία είναι εξοικειωμένοι - όπως το Prettier, το ESLint, το VS Code intellisense κ.λπ.

Και τέλος, υπάρχει η απευθείας συγγραφή WebAssembly. Με τον ίδιο τρόπο όπως οι γλώσσες assembly, η δυαδική μορφή WebAssembly έχει αναπαράσταση κειμένου - οι δύο έχουν αντιστοιχία ένα προς ένα μεταξύ τους. Αυτή η μορφή μπορεί να γραφτεί "με το χέρι" και στη συνέχεια να μετατραπεί στη δυαδική μορφή με οποιοδήποτε από τα διάφορα διαθέσιμα εργαλεία WebAssembly text-to-binary. Η αναπαράσταση κειμένου της WebAssembly αναλύεται στο επόμενο κεφάλαιο.

2.4: Μορφές & Γλώσσες WebAssembly

Η WebAssembly έχει μια δυαδική μορφή και μια μορφή κειμένου. Η δυαδική μορφή (με επέκταση αρχείου .wasm) είναι η συμπαγής δυαδική μορφή εντολών για ένα stack-based VM και η μορφή κειμένου (με επέκταση αρχείου .wat) είναι μια μορφή αναγνώσιμη από τον άνθρωπο που έχει σχεδιαστεί για να βοηθήσει τους προγραμματιστές να δουν τον πηγαίο κώδικα ενός module WebAssembly. Η μορφή κειμένου μπορεί επίσης να χρησιμοποιηθεί για τη συγγραφή κώδικα που μπορεί να μεταγλωττιστεί στη δυαδική μορφή. Παρακάτω παρατίθεται ένα παράδειγμα μιας απλής συνάρτησης πρόσθεσης γραμμένης σε μορφή κειμένου:

```
(module
  (func $add
    (param $a i32)
    (param $b i32)
    (result i32)
    local.get $a local.get $b i32.add
  )
  (export "add" (func $add))
)
```

Πριν ξεκινήσει η επεξήγηση αυτού του παραδείγματος, πρέπει να επισημανθεί ξανά ότι, τόσο στη δυαδική μορφή όσο και στη μορφή κειμένου, η θεμελιώδης μονάδα κώδικα της WebAssembly είναι το module. Στη μορφή κειμένου, ένα module αναπαρίσταται ως ένα μεγάλο S-expression (symbolic expression). Το S-expression είναι μια πολύ παλιά και πολύ απλή μορφή κειμένου για την αναπαράσταση δέντρων, και έτσι ένα module μπορεί

να θεωρηθεί ως ένα δέντρο κόμβων που περιγράφουν τη δομή και τον κώδικά του. Αρχικά ως γίνει η αρχή με το απλούστερο, συντομότερο δυνατό module WebAssembly:

```
(module)
```

Αυτό το module είναι εντελώς κενό, αλλά εξακολουθεί να είναι ένα έγκυρο module. Αντιπροσωπεύει μόνο τον κόμβο ρίζας "module" από μόνο του. Το ενδιαφέρον μέρος έρχεται στη συνέχεια. Όλος ο κώδικας σε μια ενότητα WebAssembly ομαδοποιείται σε συναρτήσεις, οι οποίες έχουν την ακόλουθη δομή ψευδοκώδικα:

```
(func <signature> <locals> <body>)
```

- Το signature δηλώνει τι δέχεται (παράμετροι) και τι επιστρέφει (τιμές επιστροφής) η συνάρτηση.
- Τα locals είναι όπως οι μεταβλητές στις περισσότερες γλώσσες προγραμματισμού, με ρητά δηλωμένους τύπους.
- Το σώμα είναι απλώς μια λίστα εντολών χαμηλού επιπέδου.

Έτσι, αυτή η δομή είναι παρόμοια με τις συναρτήσεις σε άλλες γλώσσες, ακόμα και αν φαίνεται διαφορετική επειδή είναι γραμμένη με S-expressions.

Το signature είναι μια ακολουθία δηλώσεων τύπων παραμέτρων ακολουθούμενη από μια λίστα δηλώσεων τύπων επιστροφής. Αξίζει να σημειωθεί εδώ ότι:

- Η απουσία ενός (result) σημαίνει ότι η συνάρτηση δεν επιστρέφει τίποτα.
- Στην αρχή, ήταν δυνατό να υπάρχει το πολύ 1 τύπος επιστροφής, αλλά αργότερα αυτό επεκτάθηκε σε οποιονδήποτε αριθμό.

Στο παράδειγμα της πρόσθεσης, βλέπουμε ότι η συνάρτηση δέχεται δύο παραμέτρους, οι οποίες είναι και οι δύο ακέραιοι αριθμοί 32-bit (i32), και επιστρέφει το αποτέλεσμα ως ακέραιο αριθμό 32-bit. Δεν δηλώνει καμία τοπική μεταβλητή, οπότε το τμήμα <locals> της συνάρτησης είναι κενό. Το σύμβολο του δολαρίου χρησιμοποιείται για να δηλώσει την ονομασία μιας μεταβλητής ή συνάρτησης:

```
(func $add (param $a i32) (param $b i32) (result i32) ... )
```

Για να εκτελέσουμε την πρόσθεση, χρησιμοποιούμε τη λέξη-κλειδί `local.get` και καλούμε τον τελεστή πρόσθεσης με την εντολή `i32.add`:

```
( ... local.get $a local.get $b i32.add )
```

Για να κατανοηθεί καλύτερα, ας γίνει αναφορά στη δομή δεδομένων μιας στοίβας (stack). Παρόλο που ο browser τη μεταγλωττίζει σε κάτι πιο αποδοτικό, η εκτέλεση της WASM ορίζεται με όρους μιας μηχανής στοίβας, όπου η βασική ιδέα είναι ότι κάθε τύπος εντολής κάνει push και/ή pop έναν συγκεκριμένο αριθμό σε/από μια στοίβα. Για παράδειγμα, η `local.get` ορίζεται να κάνει push την τιμή της `local` που διάβασε στη στοίβα, και η `i32.add` κάνει pop δύο τιμές `i32`, υπολογίζει το άθροισμά τους και έπειτα κάνει push την προκύπτουσα τιμή `i32`.

Όταν καλείται μια συνάρτηση, ξεκινάει με μια άδεια στοίβα η οποία σταδιακά γεμίζει και αδειάζει καθώς εκτελούνται οι εντολές του `<body>`. Η στοίβα στο παράδειγμα της πρόσθεσης περιέχει ακριβώς μία τιμή `i32` - το αποτέλεσμα της έκφρασης $(\$a + \$b)$, το οποίο χειρίζεται η `i32.add`. Η τιμή επιστροφής μιας συνάρτησης είναι απλώς η τελική τιμή που παραμένει στη στοίβα.

Τέλος, για να γίνει χρήσιμο αυτό το module, πρέπει να εξαχθεί. Η δήλωση εξαγωγής μοιάζει ως εξής:

```
(export "add" (func $add))
```

Εδώ, το `"add"` είναι το όνομα με το οποίο θα αναγνωρίζεται η συνάρτηση στη JavaScript ενώ το `$add` επιλέγει ποια συνάρτηση WebAssembly μέσα από το module εξάγεται.

Αυτό το βασικό παράδειγμα είναι αρκετό για μια επίδειξη, αλλά ως κατακλείδα πριν γίνει η συνέχεια, θα πρέπει να επισημανθεί ότι υπάρχουν επιλογές για πιο σύνθετες λειτουργίες, όπως η κλήση μιας συνάρτησης μέσα από μια άλλη συνάρτηση, η εισαγωγή συναρτήσεων στην WebAssembly από τη JavaScript, η χρήση global μεταβλητών, η χρήση των WebAssembly Tables, η χρήση WebAssembly threads και shared memories, κ.λπ.

Τώρα θα παρουσιαστεί πώς μοιάζει ο κώδικας WASM στη δυαδική του μορφή. Αυτό μπορεί να γίνει με το εργαλείο κονσόλας `wat2wasm2`, αλλά είναι επίσης διαθέσιμο ως

² Το `wat2wasm` είναι κομμάτι του [WABT: The WebAssembly Binary Toolkit](#).

online demo³. Σε αυτό το παράδειγμα θα χρησιμοποιηθεί αυτό το απλό πρόγραμμα που επιστρέφει μια σταθερά:

```
(module
  (func (export "fortytwo")
    (result i32)
    i32.const 42
    return))
```

Το δυαδικό αρχείο δεν μπορεί να προβληθεί με έναν κανονικό επεξεργαστή κειμένου, αλλά το εργαλείο που αναφέρθηκε προηγουμένως μπορεί να βοηθήσει προσθέτοντας μια επιλογή εντολών σε αυτό:

```
wat2wasm simple.wat -v
```

Ο κώδικας .wasm είναι οργανωμένος σε bytes και οι ίδιες οι ενότητες είναι οργανωμένες σε τμήματα (sections) όπως τα function section, type section, code section, name section και ούτω καθεξής. Ευτυχώς, υπάρχουν επίσης σχόλια που βοηθούν στην κατανόηση του κάθε byte. Τα τέσσερα πρώτα bytes αντιστοιχούν στον μαγικό αριθμό του δυαδικού WASM, ο οποίος μεταφράζεται σε \0asm και προσδιορίζει το αρχείο ως δυαδικό αρχείο .wasm. Τα επόμενα τέσσερα bytes αντιπροσωπεύουν την έκδοση WASM (η οποία είναι ακόμη η version 1):

```
; WASM_BINARY_MAGIC
00000000: 0061 736d
; WASM_BINARY_VERSION
00000004: 0100 0000
```

Το κάθε section έχει ένα αναγνωριστικό ID. Ανάλογα με το είδος των τμημάτων, χρησιμοποιούνται τα ακόλουθα αναγνωριστικά:

ID	Section
0	custom section
1	type section
2	import section
3	function section

³ [wat2wasm demo](#)

4	table section
5	memory section
6	global section
7	export section
8	start section
9	element section
10	code section
11	data section
12	data count section

Το πρώτο byte κάθε τμήματος αντιπροσωπεύει το αναγνωριστικό τμήματος και το μέγεθός του (σε bytes):

```
; section code (ID)
0000008: 01
; section size (5 bytes)
0000009: 05
```

Το type section περιέχει υπογραφές συναρτήσεων. Το παράδειγμα που χρησιμοποιείται έχει μία συνάρτηση με μηδέν παραμέτρους και ένα αποτέλεσμα επιστροφής τύπου i32:

```
; num types (number of types)
00000a: 01
; func
00000b: 60
; num params (number of parameters)
00000c: 00
; num results (number of results)
00000d: 01
; i32 (result type)
00000e: 7f
```

Το επόμενο τμήμα, function, αποθηκεύει δείκτες της υπογραφής της συνάρτησης:

```
; section code
000000f: 03
; section size
0000010: 02
; num functions (number of functions)
0000011: 01
; function 0 signature index
0000012: 00
```

Στη συνέχεια, ακολουθεί το τμήμα export. Το τμήμα ορίζει τον αριθμό των εξαγωγών, καθώς και το όνομα και το είδος της εξαγωγής μαζί με έναν δείκτη σε αυτόν:

```
; section code
0000013: 07
; section size
0000014: 0c
; num exports (number of exports)
0000015: 01
; string length (length of the name)
0000016: 08
; export name (fortytwo)
0000017: 666f 7274 7974 776f
; export kind (0 for function)
000001f: 00
; export func index
0000020: 00
```

Το τμήμα code που ακολουθεί αντιπροσωπεύει τον πραγματικό κώδικα της συνάρτησης. Το παράδειγμα που χρησιμοποιείται σπρώχνει μια αριθμητική σταθερά i32 στη στοίβα, η οποία επιστρέφεται ως αποτέλεσμα:

```
; section code
0000021: 0a
; section size
0000022: 06
```

```

; num functions
0000023: 01
; func body size
0000024: 04
; local decl count
0000025: 00
; i32.const (refers to the instruction)
0000026: 41
; i32 literal (number 42)
0000027: 2a
; end
0000028: 0b

```

Το τμήμα με το αναγνωριστικό 0 ονομάζεται custom section. Τα custom τμήματα προορίζονται να χρησιμοποιηθούν για πληροφορίες εντοπισμού σφαλμάτων ή για επεκτάσεις τρίτων κατασκευαστών και αγνοούνται από τη σημασιολογία WebAssembly. Τα περιεχόμενά τους αποτελούνται από ένα όνομα που προσδιορίζει περαιτέρω το προσαρμοσμένο τμήμα, ακολουθούμενο από μια μη ερμηνευμένη ακολουθία bytes για προσαρμοσμένη χρήση.

Όσον αφορά τις γλώσσες που χρησιμοποιούν την WebAssembly ως στόχο μεταγλώττισης, υπάρχει ήδη μια μεγάλη ποικιλία γλωσσών που μπορούν να μεταγλωττιστούν σε WASM, ακόμη και σε πειραματικό επίπεδο. Η συμμόρφωση των γλωσσών καθορίζεται σε μεγάλο βαθμό από την υποστήριξη από τους μεταγλωττιστές και έχουν αναφερθεί περίπου 40 γλώσσες προγραμματισμού που υποστηρίζουν την WebAssembly ως στόχο μεταγλώττισης. Περιλαμβάνονται οι περισσότερες από τις δημοφιλείς γλώσσες, αλλά παρακάτω παρατίθενται αυτές που έχουν τουλάχιστον beta ή σταθερή υποστήριξη:

- **C/C++:** Δύο από τα πιο δημοφιλή εργαλεία για τη μεταγλώττιση κώδικα C και C++ σε WebAssembly είναι το Emscripten και το Cheerp. Το Emscripten είναι ένας μεταγλωττιστής LLVM σε JavaScript ή/και WebAssembly. Παίρνει κώδικα bit της LLVM και τον μεταγλωττίζει σε JavaScript ή WASM. Το Cheerp είναι ένας εμπορικός μεταγλωττιστής C/C++ ανοικτού κώδικα που ισχυρίζεται ότι μπορεί να μεταγλωττίσει σχεδόν οποιονδήποτε κώδικα C/C++ (μέχρι C++14) σε WebAssembly, JavaScript, asm.js ή σε συνδυασμό αυτών.
- **Rust:** Η Rust μπορεί να μεταγλωττίσει σε WebAssembly εγγενώς ή σχεδόν εγγενώς χρησιμοποιώντας ένα εύχρηστο εργαλείο που ονομάζεται wasm-pack

(διαθέσιμο από το cargo, το package manager της Rust) και στοχεύοντας την Wasm κατά τη μεταγλώττιση (που ονομάζεται "wasm32-unknown-unknown" αν ο μεταγλωττιστής Rust χρησιμοποιείται από μόνος του ή απλά "web" αν χρησιμοποιείται το wasm-pack). Η Rust είναι από τις πιο αγαπητές γλώσσες προγραμματισμού για Wasm και υπάρχουν επίσης πολλά ακόμη εργαλεία που υποστηρίζονται από την κοινότητα.

- **C#/.Net:** Η χρήση της C# στο διαδίκτυο γίνεται δυνατή με το Blazor, ένα κομμάτι του ASP.NET (το οποίο με την σειρά του αποτελεί επέκταση της πλατφόρμας ανάπτυξης .NET της Microsoft). Κατασκευάζει διαδραστικά και επαναχρησιμοποιήσιμα UI components που κατασκευάζονται με τη χρήση C#, HTML και CSS. Τόσο ο κώδικας του client όσο και ο κώδικας του server είναι γραμμένος σε C#, επιτρέποντας στον προγραμματιστή να μοιράζεται κώδικα και libraries.
- **AssemblyScript:** Η AssemblyScript μεταγλωττίζει μια γλώσσα που μπορεί να θεωρηθεί υποσύνολο της TypeScript σε WebAssembly χρησιμοποιώντας ένα εργαλείο που ονομάζεται Binaryen. Αντί να απαιτείται η εγκατάσταση πολύπλοκων toolchains, η AssemblyScript μπορεί να εγκατασταθεί εύκολα με μια απλή εντολή npm (το package manager του Node.js).
- **Go:** Όπως και η Rust, η Go υποστηρίζει εγγενώς τη μεταγλώττιση σε WebAssembly καθορίζοντάς την ως στόχο μεταγλώττισης (με την επιλογή GOARCH=wasm).
- **Java:** Για τη Java υπάρχουν τρία εργαλεία, που ονομάζονται TeaVM, JWebAssembly και CheerpJ, τα οποία μεταγλωττίζουν το Java bytecode σε JavaScript και WebAssembly. Επειδή χρησιμοποιούν αρχεία κλάσεων Java ως είσοδο, ορισμένα από αυτά μπορούν να υποστηρίξουν και άλλες γλώσσες που μεταγλωττίζονται σε bytecode Java, όπως η Clojure, η Groovy, η JRuby, η Jython, η Kotlin και η Scala.
- **Python:** Για τη χρήση της Python με την WebAssembly, υπάρχει ένα εργαλείο που ονομάζεται Pyodide. Αυτό παρέχει μια μεταφορά του πιο δημοφιλούς runtime της Python, το CPython, στη WebAssembly μέσω του Emscripten. Το function interface του του επιτρέπει την ανάμειξη Python και JavaScript μέσα στον κώδικα, ενώ περιλαμβάνει επίσης πλήρη υποστήριξη για χειρισμό σφαλμάτων, async/await και πολλά άλλα. Ένα άλλο ενδιαφέρον εργαλείο είναι το RustPython, ένας διερμηνέας Python γραμμένος σε Rust. Δεδομένου ότι η Rust μεταγλωττίζει σε WebAssembly, το εργαλείο αυτό καταφέρνει να προσφέρει υποστήριξη για τη χρήση της Python στον ιστό.

Όπως υποδηλώνει το όνομά της, η WebAssembly έχει σχεδιαστεί για να εκτελείται μέσω του διαδικτύου σε browsers. Αλλά έχει τη δυνατότητα για ένα τεράστιο μέλλον εκτός

διαδικτύου χάρη στο WASI, το WebAssembly System Interface. Αυτή η διεπαφή συστήματος επιτρέπει την εκτέλεση της WebAssembly τόσο εκτός όσο και εντός του ιστού. Για την εκτέλεση της Wasm εκτός του ιστού με τη χρήση του WASI χρησιμοποιούνται αυτόνομα runtimes της Wasm που μπορούν να λειτουργήσουν ως διερμηνείς. Η υπόσχεση της χρήσης της σε περιβάλλοντα εκτός του web (με ή χωρίς JavaScript VM όπως το Node.js) εκτείνεται από περιβάλλοντα δοκιμών με χρήση βασικών κελυφών (shells) έως εμπορικές εφαρμογές για συσκευές IoT, εφαρμογές για κινητά ή υπολογιστές, διακομιστές κέντρων δεδομένων ή ως εκτελέσιμο αρχείο μέσα σε μεγαλύτερα προγράμματα.

Τα WebAssembly runtimes παρέχουν ουσιαστικά τη δυνατότητα κλήσης του WebAssembly VM. Η υλοποίηση κάθε runtime διαφέρει, αλλά συνήθως το runtime επιτρέπει τη μεταβίβαση της δυαδικής μορφής του WebAssembly στο VM κατά το instantiation του VM. Το runtime επιτρέπει επίσης στον καλούντα να καλέσει μια συνάρτηση (καθώς και να περάσει τυχόν απαιτούμενες παραμέτρους της συνάρτησης). Το WebAssembly runtime επίσης παρέχει έναν μηχανισμό για την επιστροφή του αποτελέσματος της κλήσης της συνάρτησης στον καλούντα. Τέλος, μόλις ολοκληρωθεί η εκτέλεση, το WebAssembly runtime είναι σε θέση να τερματίσει το εφήμερο VM instance. Υπάρχουν αρκετά runtimes κατάλληλα για χρήση αυτή τη στιγμή και ο κάθε υλοποιητής Wasm runtime μπορεί να προσφέρει διαφορετικά επίπεδα πόρων και επικοινωνίας μεταξύ του VM και του καλούντος. Μερικά από τα πιο δημοφιλή runtimes αυτή την στιγμή παρατίθενται παρακάτω:

- **Wasmer:** Το κύριο WebAssembly runtime, έχει την καλύτερη συνολική υποστήριξη για διάφορες γλώσσες προγραμματισμού και η ανάπτυξή του είναι πολύ ενεργή.
- **Wasmtime:** Το Wasmtime υποστηρίζει λιγότερες γλώσσες, αλλά είναι γρήγορο, ασφαλές και προσφέρει πολλές δυνατότητες παραμετροποίησης.
- **WasmEdge:** Ένα ελαφρύ, υψηλής απόδοσης και επεκτάσιμο WebAssembly runtime. Το WasmEdge είναι ένα επίσημο έργο sandbox που φιλοξενείται από το Cloud Native Computing Foundation (CNCF). Είναι επίσης το ταχύτερο Wasm VM σήμερα σύμφωνα με το IEEE.
- **WebAssembly Micro Runtime (WAMR):** Το WAMR είναι ένα runtime που εξειδικεύεται στο να χρησιμοποιεί μικρό footprint.

Ένα σημαντικό στοιχείο που πρέπει να επισημανθεί είναι ότι, επειδή η WebAssembly είναι μια σχετικά νέα (και αναμφισβήτητα επαναστατική) τεχνολογία, το οικοσύστημά της είναι ακόμη ασταθές. Νέα εργαλεία μπορούν να εμφανιστούν και τα παλιά μπορεί να χάσουν το ενδιαφέρον των κοινοτήτων τους ή να εγκαταλειφθούν. Παρόλα αυτά, τα έργα που αναφέρονται ονομαστικά στα κεφάλαια αυτής της εργασίας φαίνεται ότι θα

διατηρήσουν το ενδιαφέρον που έχουν προκαλέσει και θα συνεχίσουν να δέχονται συνεχή ανάπτυξη όσο ωριμάζει η νέα τεχνολογία.

3ο Κεφάλαιο:

Μια πρακτική προσέγγιση: Επεξεργασία Εικόνας

3.1: Τι είναι η επεξεργασία εικόνας και γιατί η WebAssembly ταιριάζει για αυτόν τον σκοπό

Η ψηφιακή επεξεργασία εικόνας (digital image processing), συχνά συντομογραφούμενη ως "επεξεργασία εικόνας", είναι ένας τομέας της ανάπτυξης λογισμικού που ασχολείται με τον χειρισμό εικόνων. Πρόκειται ουσιαστικά για τη χρήση διαφόρων αλγορίθμων για την εξαγωγή πληροφοριών σχετικά με τις εικόνες ή την τροποποίησή τους σε πιο χρήσιμες μορφές. Για παράδειγμα, όταν μια φωτογραφική μηχανή τραβάει μια εικόνα, κάνει τη δική της επεξεργασία στην εικόνα για την τελική προβολή της από τον χρήστη. Ή ενώ ένα αυτόνομο όχημα βρίσκεται σε κίνηση, η επεξεργασία εικόνας χρησιμοποιείται για την επεξεργασία των εισερχόμενων δεδομένων βίντεο από το περιβάλλον, συνήθως για την ανίχνευση ή/και παρακολούθηση αντικειμένων.

Ανάλογα με το μέγεθος της εικόνας ή τον ρυθμό μετάδοσης bit (bitrate) της ροής βίντεο, αυτοί οι αλγόριθμοι μπορεί να γίνουν αρκετά υπολογιστικά εντατικοί, γι' αυτό και η WebAssembly είναι μια εξαιρετική επιλογή για εφαρμογές τέτοιου είδους, αν αυτές έχουν κατασκευαστεί για την πλατφόρμα του ιστού. Δεδομένου ότι είναι γενικά ταχύτερη και προσφέρει πιο σταθερές και προβλέψιμες αποδόσεις από τη JavaScript, η WebAssembly επιτρέπει την ανάπτυξη εφαρμογών ιστού που μπορούν να αποδώσουν ακόμη και σε κοινές κινητές συσκευές ή την ανάπτυξη εφαρμογών ιστού για desktop υπολογιστές που επιτρέπουν πιο σύνθετες λειτουργίες. Τέλος, για μια εφαρμογή, το να κάνει απλώς τη δουλειά της δεν είναι συνήθως αρκετό. Είναι επιθυμητό να υπάρχει μια responsive εμπειρία χρήσης που δεν θα δυσαρεστήσει τον χρήστη. Για μια εφαρμογή, το μοντέλο επιδόσεων RAIL της Google απαιτεί σε ιδανική περίπτωση να απεικονίζεται ένα frame κάθε 16ms το λιγότερο (για 60 frames ανά δευτερόλεπτο)⁴, δηλαδή το πολύ 10ms για την εφαρμογή να παράγει το frame επειδή χρειάζονται 6ms για την απεικόνιση από τον browser. Όμως ειδικά σε περιπτώσεις όπου ένα frame χρειάζεται πάνω από

⁴ [“Measure performance with the RAIL model”](#)

100ms για να απεικονιστεί, ένας χρήστης μπορεί να διαπιστώσει μία αποσύνδεση της ενέργειας από το αποτέλεσμα της.

Για το αντικείμενο αυτής της πτυχιακής εργασίας, η εστίαση θα παραμείνει στο image editing, μια από τις μεγαλύτερες εφαρμογές της επεξεργασίας εικόνας. Αυτό θεωρείται μια δημιουργική, καλλιτεχνική πράξη και γίνεται για την αφαίρεση ανεπιθύμητων στοιχείων, όπως κηλίδες σκόνης και γρατσουνιές, την προσαρμογή της γεωμετρίας της εικόνας, όπως περιστροφή και περικοπή, τη διόρθωση των εκτροπών του φακού, την αλλαγή του χρώματος ή την προσθήκη ειδικών εφέ στην εικόνα. Συχνά οι εργασίες που αφορούν την επεξεργασία εικόνας είναι επαναλαμβανόμενες και χρειάζονται έντονη επεξεργασία.

Πριν από την WebAssembly, για την ανάπτυξη ενός web app για τον σκοπό του image editing (στον client), η μόνη επιλογή του προγραμματιστή για τον προγραμματισμό της λογικής της εφαρμογής ήταν η χρήση JavaScript. Αυτή τη στιγμή, υπάρχει πληθώρα εφαρμογών και βιβλιοθηκών JavaScript στο διαδίκτυο που έχουν δημιουργηθεί για αυτόν τον σκοπό και ως παράδειγμα επιλέχθηκαν δύο δημοφιλή έργα ανοικτού κώδικα:

- Το ένα από αυτά ονομάζεται CamanJS⁵ και είναι μια βιβλιοθήκη που προσφέρει image editing με τον καμβά HTML5 και τη JavaScript. Πρόκειται για μια γνωστή και ισχυρή βιβλιοθήκη που προσφέρει τόσο μεγάλη ποικιλία ενσωματωμένων λειτουργιών όσο και επεκτασιμότητα με plugins. Είναι εντελώς ανεξάρτητο από άλλες βιβλιοθήκες και λειτουργεί τόσο στο NodeJS όσο και στον browser.
- Το άλλο ονομάζεται glfx.js⁶ και στοχεύει στη χρήση της κάρτας γραφικών της συσκευής με τη τεχνολογία WebGL για τη δημιουργία εφέ όπως το blurring και το wrapping.

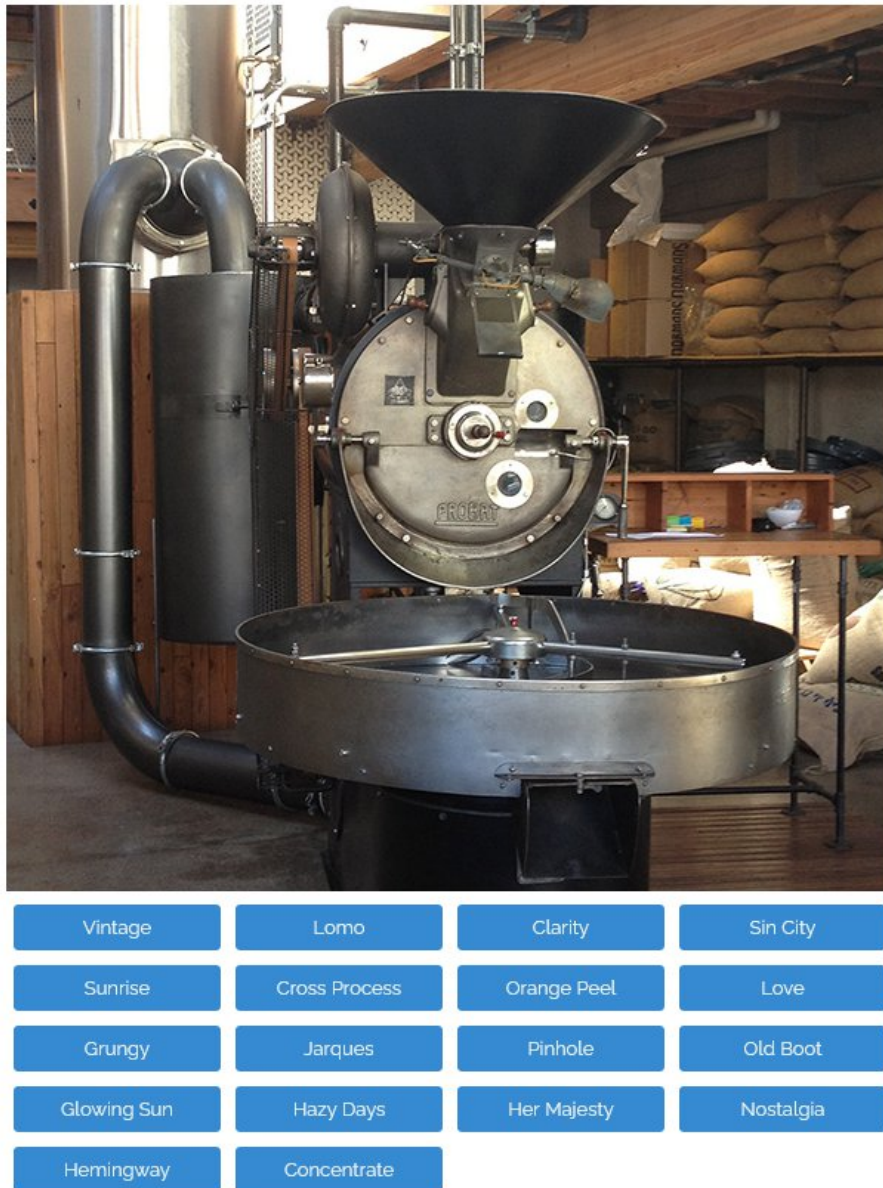
Αυτά τα παραδείγματα επιλέχθηκαν επίσης επειδή προσφέρουν ζωντανή επίδειξη των δυνατοτήτων τους στις ιστοσελίδες των έργων τους. Σε ένα γρήγορο υπολογιστή desktop, ένα φίλτρο, ένα θόλωμα ή μια λειτουργία διόρθωσης εικόνας ανταποκρίνεται σχεδόν άμεσα. Εάν η ίδια λειτουργία εκτελεστεί σε μια κοινή κινητή συσκευή όπως ένα smartphone, μπορεί ήδη να παρατηρηθεί μια μικρή καθυστέρηση κατά την εκτέλεση της λειτουργίας. Αυτή η καθυστέρηση μπορεί μόνο να επιδεινωθεί αισθητά όταν χρησιμοποιείται μια μεγαλύτερη εικόνα, όπως μια εικόνα Full HD (με ανάλυση 1920x1080) ή μια συνηθισμένη φωτογραφία από smartphone 12MP (με ανάλυση 4000x3000).

⁵ [Παραδείγματα CamanJS](#) και [αποθετήριο κώδικα](#)

⁶ [Παραδείγματα glfx.js](#) και [αποθετήριο κώδικα](#)

Preset Examples

CamanJS has a number of preset filters that show off how to generate consistent effects on many types of images.

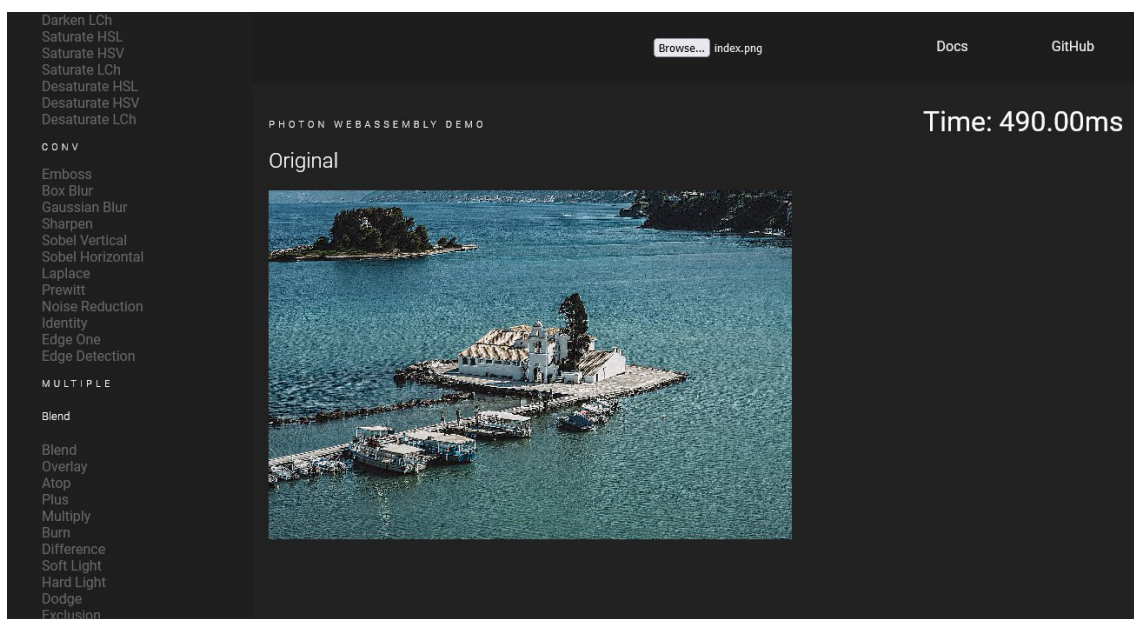
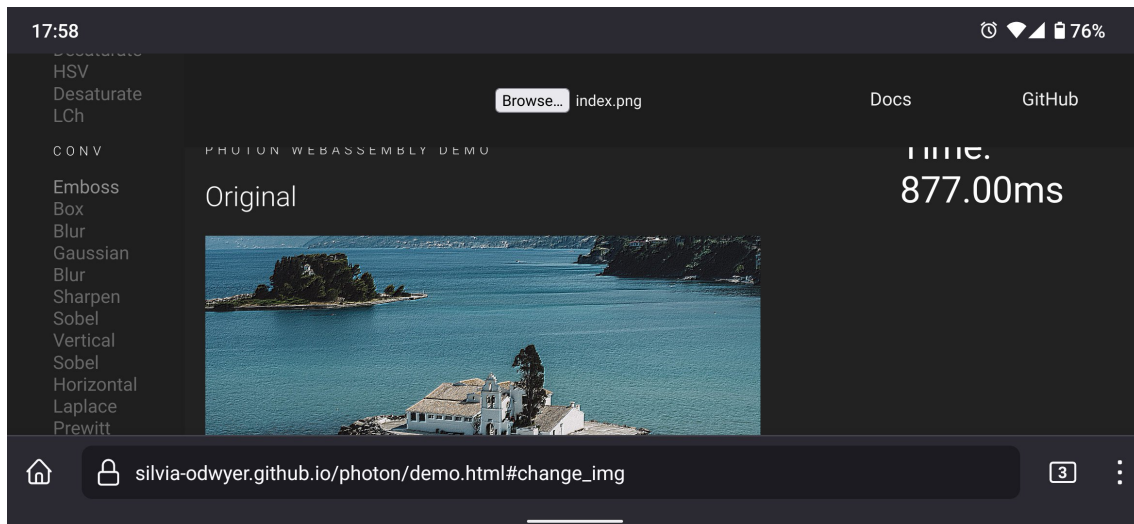


Στιγμιότυπο 2: Το CamanJS δίνει μια επίδειξη των φίλτρων που μπορεί να παρέχει. Προτείνονται δοκιμές με το φίλτρο “Concentrate” σε διαφορετικές συσκευές για να παρατηρηθεί η διαφορά σε χρόνο επεξεργασίας.

Μετά την κυκλοφορία της WebAssembly, ήταν αναπόφευκτο ότι η τεχνολογία θα χρησιμοποιούνταν για την προσπάθεια δημιουργίας responsive web εφαρμογών για image editing, μιας και αυτές είναι δημοφιλείς και χρησιμοποιούνται στις περισσότερες συσκευές και πλατφόρμες, από επιτραπέζιους υπολογιστές έως smartphones και tablet.

Μια από τις πιο δημοφιλείς προσπάθειες μέχρι σήμερα είναι το Photon, μια βιβλιοθήκη για image editing που κατασκευάστηκε για την WebAssembly με χρήση της γλώσσας Rust. Διαθέτει μια μεγάλη ποικιλία λειτουργιών, γρήγορες ταχύτητες, προσφέρει απλότητα στην χρήση με κλήση συναρτήσεων μόνο με JavaScript, καθώς και τη δυνατότητα χρήσης της σε Node.js ή σε περιβάλλοντα της Rust. Πρόκειται για ένα έργο ανοιχτού κώδικα που είναι αρκετά δημοφιλές στο GitHub, υποστηριζόμενο από ευανάγνωστο documentation και μια ενεργή κοινότητα.

Ακριβώς όπως και τα έργα από τα παραδείγματα JavaScript, το Photon προσφέρει ζωντανή επίδειξη των δυνατοτήτων του στην ιστοσελίδα του έργου. Για καλύτερη επίδειξη, συνιστάται η μεταφόρτωση μιας μεγαλύτερης εικόνας για δοκιμή. Σε έναν desktop υπολογιστή, μπορεί να παρατηρηθεί ότι όλες οι λειτουργίες ανταποκρίνονται άμεσα όταν χρησιμοποιούνται και, μπορεί να παρατηρηθεί ότι ακόμη και όταν χρησιμοποιείται μια μεγάλη εικόνα σε μια κινητή συσκευή, σχεδόν όλες οι λειτουργίες είναι τουλάχιστον χρησιμοποιήσιμες και δεν χρειάζονται περισσότερο από 1-2 δευτερόλεπτα για να ολοκληρωθεί η επεξεργασία.



Στιγμιότυπα 3 και 4: Δοκιμές του φίλτρου “Emboss” του Photon σε κινητή συσκευή και σε σταθερό υπολογιστή. Ακόμα και σε φωτογραφία ανάλυσης 2400x1600, μπορούν να παρατηρηθούν ικανοποιητικές αποδόσεις για τα δεδομένα των συσκευών.

Για την περαιτέρω κατανόηση του τρόπου λειτουργίας και χρήσης της WebAssembly σε αυτόν τον τομέα, στα πλαίσια της παρούσας πτυχιακής εργασίας αναπτύχθηκε μια εφαρμογή ιστού με περιορισμένη λειτουργικότητα. Αυτό το κεφάλαιο επικεντρώνεται σε μια γενικά πιο πρακτική ανάλυση του πώς μπορεί να χρησιμοποιηθεί η WebAssembly, τι εργαλεία υπάρχουν και ποιά χρησιμοποιήθηκαν, ποιά ήταν η μεθοδολογία ανάπτυξης και τι δοκιμές έγιναν. Επίσης δίνονται διάφορες εξηγήσεις σχετικά με την εγκατάσταση, τη χρήση και άλλες πτυχές του έργου, καθώς και την έμπνευση για το έργο και τα εμπόδια που συναντήθηκαν κατά τη διάρκεια της διαδικασίας ανάπτυξης.

3.2: Εισαγωγή στην εφαρμογή

Η εφαρμογή που αναπτύχθηκε είναι απλή στη χρήση και άμεσα διαθέσιμη σε οποιαδήποτε πλατφόρμα στην οποία παρέχεται browser που να υποστηρίζει το WebAssembly και έχει δοκιμαστεί ότι λειτουργεί σε γνωστά προγράμματα περιήγησης, όπως ο Chrome και ο Firefox, τόσο σε desktops όσο και σε smartphone Android. Υποστηρίζει ορισμένες απαραίτητες λειτουργίες, όπως η λήψη (downloading), η μεταφόρτωση (uploading) εικόνων και η επαναφορά εικόνας (reseting). Λειτουργικά, μπορεί να κάνει κάποιους βασικούς χειρισμούς όπως το καθρέφτισμα εικόνας (mirroring), αύξηση/μείωση της φωτεινότητας ή του κορεσμού, αντιστροφή των χρωμάτων της εικόνας ή εφαρμογή φίλτρων όπως είναι το φίλτρο ασπρόμαυρου. Η εφαρμογή παρέχεται ως μια στατική σελίδα που φιλοξενείται στο GitHub, χωρίς καθόλου backend κώδικα.

Όπως έχει ήδη αναφερθεί, υπάρχουν πολλά διαφορετικά είδη πιθανών εφαρμογών της WebAssembly. Ωστόσο, η επεξεργασία εικόνων επιλέχθηκε επειδή ήταν αρκετά απλή για να γίνει η αρχή από προγραμματιστή χωρίς κάποια εξειδικευμένη εμπειρία, ενώ παράλληλα μπορεί να δημιουργήσει κάτι που είναι διαδραστικό και παράγει ένα άμεσο οπτικό αποτέλεσμα. Η προσπάθεια εμπνεύστηκε επίσης σε μεγάλο βαθμό από προηγούμενα έργα όπως το Photon, και ακόμη από την ιδέα ότι στο σύντομο μέλλον ίσως να μπορεί να είναι δυνατή η χρήση ενός πλήρως εξοπλισμένου λογισμικού επεξεργασίας εικόνας (όπως το Adobe Photoshop) κατευθείαν στον browser.

3.3: Εργαλεία και τεχνολογίες που χρησιμοποιήθηκαν

Οι λειτουργίες της εφαρμογής γράφτηκαν σε AssemblyScript, μια γλώσσα που επιλέχθηκε για το οικείο συντακτικό της που μοιάζει με την TypeScript. Παρά την ομοιότητά τους, η AssemblyScript χρησιμοποιεί αποκλειστικά τους τύπους μεταβλητών της WebAssembly (όπως για παράδειγμα ο i32 για ακέραιους αριθμούς 32-bit ή ο f64 για αριθμούς κινητής υποδιαστολής 64-bit) και πρέπει να μεταγλωττιστεί σε κώδικα μηχανής πριν να μπορεί να χρησιμοποιηθεί (ahead of time compilation). Επίσης η AssemblyScript παρέχει μια βασική βιβλιοθήκη που μοιάζει με αυτή της JavaScript, με πολλές από τις κλάσεις και τα namespaces που έχουν συνηθιστεί από τη JavaScript, συμπεριλαμβανομένων των Math, Mathf, Array, String, Map, Date, Number και άλλα. Πρέπει όμως να σημειωθεί ότι παρόλο που είναι επηρεασμένες σε μεγάλο βαθμό από τις κλάσεις της JavaScript, αυτές δεν είναι αντίστοιχες μία προς μία και δεν έχουν περαστεί όλες οι δυνατότητες στις βιβλιοθήκες της AssemblyScript, είτε λόγω της φύσης της WebAssembly ή επειδή απλά δεν έχουν υλοποιηθεί σε αυτό τον χρόνο.

Το AssemblyScript χρησιμοποιεί επίσης εκτενώς το οικοσύστημα Node.js. Για να δημιουργηθεί ένα νέο έργο, πρέπει να εκτελεστούν μερικές εντολές διαχείρισης πακέτων και μεταγλωττιστή node για να δημιουργηθεί ένα έργο node, να εγκατασταθεί ο μεταγλωττιστής της AssemblyScript και να ρυθμιστεί η δομή των φακέλων και των αρχείων για ένα νέο έργο AssemblyScript:

```
npm init
npm install --save-dev assemblyscript
npx asinit .
```

Με τον ίδιο τρόπο μπορούν να εγκατεστηθούν όποια ακόμη εργαλεία χρειαστούν ή είναι επιθυμητά. Ένα τέτοιο παράδειγμα στην εργασία είναι το πακέτο `ws`⁷, ένα πρόγραμμα που χρησιμοποιήθηκε για την δημιουργία ενός τοπικού http server για τις δοκιμές του κώδικα.

Φυσικά, η JavaScript έπρεπε επίσης να χρησιμοποιηθεί για την δημιουργία instance της WebAssembly (δηλαδή να την “καλέσει”), για την κλήση των συναρτήσεων που εξάγει αυτή, για την δέσμευση και διαχείριση της κοινής μνήμης μεταξύ των δύο, και για να χειριστεί λειτουργίες που δεν σχετίζονται με την επεξεργασία της εικόνας, όπως η λήψη και η φόρτωση της επεξεργασμένης εικόνας. Επίσης χειρίζεται την γενική διαδραστικότητα της εφαρμογής με event listener, όπως για παράδειγμα την χρήση των διάφορων κουμπιών της διεπαφής χρήστη.

Η ίδια η εικόνα εμφανίζεται σε ένα HTML στοιχείο `<canvas>`, κάτι που συχνά χρησιμοποιείται για την εμφάνιση γραφικών. Μια προεπιλεγμένη εικόνα φορτώνεται αρχικά στον καμβά και οι επόμενες λειτουργίες σχεδιάζουν πάνω της χρησιμοποιώντας τα δεδομένα από τη μνήμη.

Τέλος, για να γίνει η εφαρμογή responsive σε όλα τα μεγέθη οθόνης και ταυτόχρονα να μην δαπανηθεί πολύς χρόνος για το styling, χρησιμοποιήθηκε το Tailwind CSS⁸, ένα CSS framework φτιαγμένο για την ταχεία κατασκευή responsive διεπαφών χρήστη. Αυτό χρησιμοποιεί κλάσεις HTML για την εφαρμογή μορφοποιήσεων στο έγγραφο χωρίς να χρειάζεται να γραφτούν αρχεία CSS από τον προγραμματιστή. Για παράδειγμα, για να δημιουργηθεί ένα κουμπί με συγκεκριμένη εμφάνιση, θα μπορούσε να χρησιμοποιηθούν αυτές η κλάσεις:

⁷ [ws: a Node.js WebSocket library](#)

⁸ [Tailwind CSS](#)

```
<button class="h-10 px-6 font-semibold rounded-md bg-black text-white" type="submit">Grayscale</button>
```

Αυτή η γραμμή διαθέτει τις ακόλουθες κλάσεις:

- **h10:** Θέτει το κουμπί σε σταθερό ύψος 10 μονάδων.
- **px-6:** Θέτει το οριζόντιο γέμισμα του κουμπιού σε 6 μονάδες.
- **font-semibold:** Ορίζει το βάρος γραμματοσειράς του κουμπιού σε semibold.
- **rounded-md:** Ορίζει την ακτίνα περιγράμματος του κουμπιού ώστε να έχει στρογγυλεμένες γωνίες.
- **bg-black:** Θέτει το χρώμα φόντου του κουμπιού σε μαύρο.
- **text-white:** Ορίζει το χρώμα κειμένου του κουμπιού σε λευκό.

Η καμπύλη εκμάθησης κατά τη χρήση του Tailwind CSS συνίσταται κυρίως από την εξοικείωση με τις βοηθητικές κλάσεις του. Μόλις γίνει αυτό, είναι εύκολο να δημιουργηθούν γρήγορα και με ακρίβεια προσαρμοσμένα στοιχεία όπως αυτό το παραπάνω κουμπί.

Επίσης, για να γίνει η εφαρμογή πραγματικά εμφανίσιμη, χρησιμοποιήθηκε ένα επιπλέον πρόσθετο που ονομάζεται Tailwind Elements⁹. Αυτό παρέχει μια μεγάλη ποικιλία από προκατασκευασμένα στοιχεία UI για χρήση με το Tailwind CSS, τα οποία προσφέρουν επιπλέον λειτουργικότητα, όπως εφέ hover, ομαδοποίηση στοιχείων, πλευρικές γραμμές και κάθε είδους σύγχρονα εφέ. Η εγκατάστασή του γίνεται εύκολα με το package manager του Node.js.

Για να φτάσει η εφαρμογή στην ολοκλήρωσή της, χρησιμοποιήθηκαν διάφορες κοινωνικές πηγές υποστήριξης για παροχή γνώσεων και τεχνικών προγραμματισμού. Το έργο ξεκίνησε χωρίς προηγούμενη προσωπική εμπειρία χρήσης TypeScript ή WebAssembly, οπότε χρησιμοποιήθηκαν σε μεγάλο βαθμό τα documentation TypeScript και AssemblyScript, καθώς και η πλατφόρμα Mozilla Developer Network για εγχειρίδια πάνω σε αυτές τις τεχνολογίες. Επιπλέον, η κοινότητα της AssemblyScript λειτουργεί επίσης μια ομάδα στην κοινωνική πλατφόρμα Discord¹⁰, όπου γίνονται ανακοινώσεις σχετικά με τη γλώσσα, συζητούνται διάφορα θέματα που αφορούν την γλώσσα και τις παρεμφερές τεχνολογίες, ή μπορεί να ζητηθεί βοήθεια σε περίπτωση που κάποιος

⁹ [Tailwind Elements Quick Start](#)

¹⁰ [AssemblyScript Official Discord Community](#)

εθελοντικά επιθυμεί να βοηθήσει. Και τέλος, υπήρξε μια ποσότητα άρθρων από ανεξάρτητες προσωπικές ιστοσελίδες και ιστοσελίδων Q&A όπως το Stack Overflow στο διαδίκτυο που βοήθησαν με την εκμάθηση.

Όσον αφορά την ίδια τη διαδικασία ανάπτυξης, χρησιμοποιήθηκε ένας υπολογιστής με εγκατεστημένη μια διανομή Linux που παρέχει τις πιο σύγχρονες εκδόσεις λογισμικού που κυκλοφορούν, που ονομάζεται Arch Linux. Ο code editor που χρησιμοποιήθηκε ήταν το δημοφιλές Visual Studio Code και το κύριο browser που χρησιμοποιήθηκε για τις δοκιμές ήταν το Firefox. Όλα τα παραπάνω είναι οικεία εργαλεία (και μάλιστα όλα τους ανοιχτό λογισμικό) και κρίθηκαν πολύ κατάλληλα για την εργασία. Για επιπλέον τρόπους δοκιμών χρησιμοποιήθηκε το πρόγραμμα περιήγησης Chrome καθώς και οι κινητές εκδόσεις των προηγούμενων browser σε smartphone Android με επεξεργαστή 8 πυρήνων. Τέλος, το git και το GitHub χρησιμοποιήθηκαν για version control, τη φιλοξενία του ανοικτού κώδικα και τη φιλοξενία της ίδιας της εφαρμογής στο διαδίκτυο.

3.4: Η δομή των αρχείων

Σε αυτήν την ενότητα ακολουθεί μια επεξήγηση των αρχείων της εφαρμογής, ξεκινώντας αλφαβητικά. Ο φάκελος assembly έχει το αρχείο index.ts το οποίο είναι το αρχείο με όλα τα AssemblyScript γραμμένα μέσα σε αυτό. Έχει επίσης ένα αρχείο tsconfig.json το οποίο σχετίζεται με τον compiler AssemblyScript. Είναι ενδιαφέρον να παρατηρηθεί ότι παρόλο που η AssemblyScript είναι ξεχωριστή γλώσσα, εξακολουθεί να χρησιμοποιεί την επέκταση αρχείου TypeScript .ts όπως καθώς και ορισμένα από τα εργαλεία της.

Ο επόμενος φάκελος, build, δημιουργείται μετά τη μεταγλώττιση του κώδικα AssemblyScript. Περιέχει τα μεταγλωττισμένα δυαδικά αρχεία WebAssembly .wasm, τις αντίστοιχες μορφές κειμένου .wat, καθώς και τα παραγόμενα source maps (για σκοπούς debugging).

Ο τρίτος φάκελος, node_modules, δημιουργείται μετά την εγκατάσταση των εξαρτήσεων (dependencies) με την αντίστοιχη εντολή npm πριν την εκτέλεση της εφαρμογής. Μέσα σε αυτόν υπάρχουν όλες οι απαραίτητες εξαρτήσεις για την εφαρμογή, η καθεμία στο δικό της φάκελο, και οι περισσότερες από αυτές έχουν το δικό τους αρχείο markdown README.md που εξηγεί ποια είναι και τι κάνει.

Τέλος, υπάρχουν μερικά σημαντικά αρχεία στον ριζικό φάκελο του έργου. Το αρχείο `asconfig.json` αναφέρεται στις ρυθμίσεις του μεταγλωττιστή της `AssemblyScript`, `asc`. Υπάρχουν πολλές διαφορετικές ρυθμίσεις που μπορούν να χρησιμοποιηθούν και είναι όλες τεκμηριωμένες στο επίσημο `documentation`. Τα αρχεία που χρησιμοποιούνται για το `styling` της εφαρμογής είναι τα `index.css` και `compiled.css`, καθώς και το `tailwind.config.js`, τα οποία δημιουργήθηκαν κατά τη διαδικασία εγκατάστασης του `framework` `Tailwind CSS` και του `plugin` του `Tailwind Elements`. Τα αρχεία για την `HTML` και τη `JavaScript` είναι τα `index.html` και `index.js` αντίστοιχα. Η προεπιλεγμένη εικόνα βρίσκεται επίσης στον ριζικό φάκελο, με όνομα `default.jpg`, και υπάρχει επίσης το αρχείο `README.md` σε μορφή `markdown`. Τα αρχεία `package-lock.json` και `package.json` είναι πακέτα `Node.js`, που δημιουργούνται κατά τη διαδικασία αρχικοποίησης μιας εφαρμογής `Node` και χρησιμοποιούνται για την εγκατάσταση των εξαρτήσεων του έργου. Τέλος, υπάρχουν ο κρυφός φάκελος `.git` και το κρυφό αρχείο `.gitignore` που υποστηρίζουν τη λειτουργικότητα `version control` με το `git`.

3.5: Εξήγηση της λογικής του κώδικα

Για να εξηγήσουμε τον κώδικα, το αρχείο `index.html` θα χρησιμοποιηθεί ως σημείο εκκίνησης. Αφού ανοιχτεί το αρχείο, μπορεί να παρατηρηθεί εύκολα ότι υπάρχει ένας μεγάλος αριθμός κλάσεων για το `Tailwind CSS`, αλλά κατά τα άλλα, πρόκειται για ένα απλό αρχείο με λιγότερες από 100 γραμμές. Ξεκινώντας από το tag `<head>`, μια σημαντική λεπτομέρεια είναι ότι το attribute `"defer"` χρησιμοποιείται για τη φόρτωση του κώδικα `JavaScript`. Αυτό σημαίνει ότι το `script` εκτελείται αφού ολοκληρωθεί η φόρτωση της ίδιας της σελίδας. Αυτή η συμπεριφορά είναι επιθυμητή, διότι διαφορετικά υπάρχει κάποια πιθανότητα το `script` να προσπαθήσει να φορτώσει την προεπιλεγμένη εικόνα σε ένα στοιχείο `καμβά` που δεν υπάρχει ακόμα στο `DOM`, οδηγώντας σε έναν κενό (ή αόρατο) `καμβά`:

```
<head>
...
  <script src="index.js" defer></script>
...
</head>
```

Στο σώμα της HTML, προστέθηκαν μερικά στοιχεία Tailwind Elements (προκατασκευασμένα στοιχεία UI) για την υποστήριξη της απαραίτητης βασικής λειτουργικότητας UI. Αυτό μπορεί να φανεί στην διαμόρφωση των στοιχείων `<div>` που συνθέτουν το τμήμα της επικεφαλίδας του ιστότοπου, του στοιχείου `<input>` για το χειρισμό uploading αρχείων και των στοιχείων `<button>` για την εφαρμογή των λειτουργιών της εφαρμογής. Το ίδιο το styling διαμορφώνεται με διαισθητικά ονόματα κλάσεων (π.χ. το `"px-6"` εφαρμόζει ένα ποσό κάθετου padding) και, παρόλο που είναι εύκολο στη χρήση, καταλήγουν να "γεμίζουν" οπτικά το αρχείο HTML. Πρόκειται για έναν συμβιβασμό για να μην χρειάζεται να γραφτεί κείμενο CSS, καθώς τα δύο αρχεία `.css` στο έργο παράγονται αυτόματα από το ίδιο το framework. Για παράδειγμα, το κουμπί που είναι υπεύθυνο για το κατέβασμα της εικόνας μοιάζει κάπως έτσι:

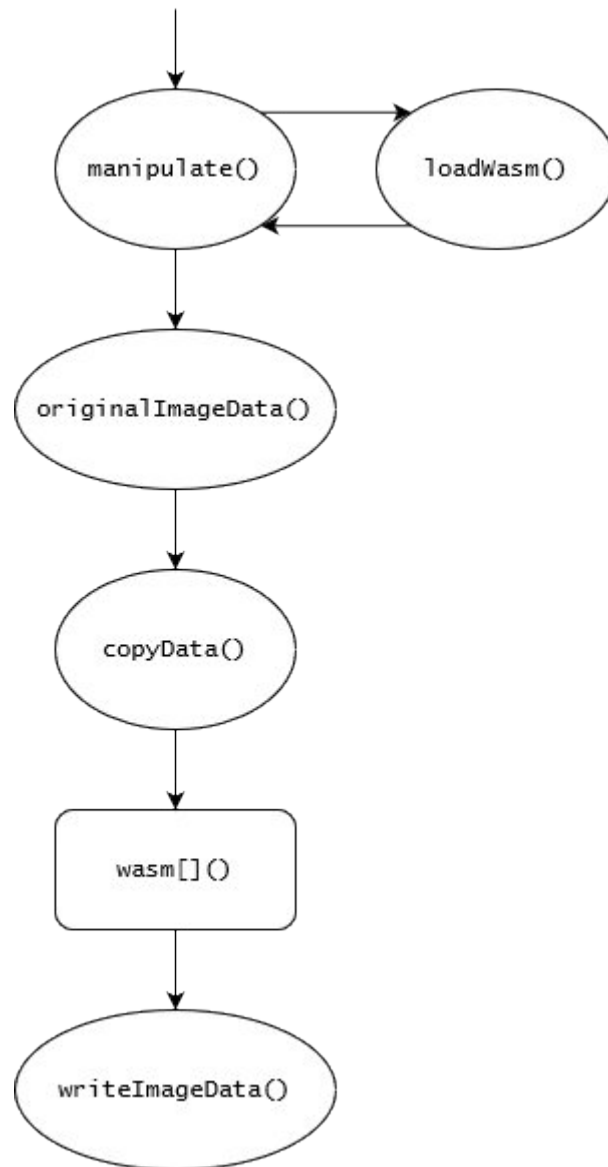
```
<button id="imageDownloader" type="button" class="inline-block
px-6 pt-2.5 pb-2 bg-blue-600 text-white font-medium text-xs
leading-normal uppercase rounded shadow-md hover:bg-blue-700
hover:shadow-lg focus:bg-blue-700 focus:shadow-lg focus:outline-
none focus:ring-0 active:bg-blue-800 active:shadow-lg transition
duration-150 ease-in-out flex align-center">
...
</button>
```

Στη συνέχεια, θα προσφερθεί μια επεξήγηση του `index.js`, του αρχείου που κρατάει όλη την λογική της JavaScript της εφαρμογής. Φιλοξενεί 8 συναρτήσεις και τουλάχιστον 13 event listeners για τη διαχείριση όλων των απαραίτητων λειτουργιών ή χειρισμών. Έχουν επίσης προστεθεί σχόλια μέσα στον κώδικα για λόγους σαφήνειας.

Οι συναρτήσεις `uploadImage(e)` και `downloadCanvas()` υποστηρίζουν τις λειτουργίες λήψης/φόρτωσης εικόνων. Για το ανέβασμα, χρησιμοποιείται ένα `FileReader` (ένας τύπος JavaScript object), ενώ για το κατέβασμα, δημιουργείται ένας προσωρινός σύνδεσμος λήψης ο οποίος "πατιέται" για τον χρήστη. Υπάρχει επίσης η συνάρτηση `original()`, η οποία επαναφέρει την εικόνα στην αρχική της κατάσταση.

Η επόμενη συνάρτηση, `manipulate(...)`, μπορεί να είναι λίγο δύσκολο να κατανοηθεί με μια πρώτη ματιά, επειδή καλεί μερικές ακόμα συναρτήσεις από το εσωτερικό της. Λαμβάνει ως παράμετρο ένα αλφαριθμητικό που υποδεικνύει το είδος της επεξεργασίας (π.χ. φίλτρο `"grayscale"`), καθώς και οποιεσδήποτε άλλες παραμέτρους που θα μπορούσαν να αναφέρονται στην ένταση της επεξεργασίας. Αρχικά λοιπόν, δημιουργεί ένα instance WASM για να φορτώσει τις εξαγωγές της, και στη συνέχεια δημιουργεί

επίσης μεταβλητές για να φορτώσει τα δεδομένα της αρχικής εικόνας καθώς και το memory buffer της WASM. Στη συνέχεια, γράφει τα δεδομένα της εικόνας στο memory buffer, εφαρμόζει την επιλεγμένη επεξεργασία και στη συνέχεια γράφει στον καμβά από την μνήμη. Η κλήσεις των συναρτήσεων ακολουθούν την παρακάτω σειρά:



Για τη δημιουργία ενός instance WASM χρησιμοποιείται η συνάρτηση loadWasm(). Στον κώδικα, συνήθως το αντιπροσωπευτικό αντικείμενο της μνήμης δεν χρειάζεται να δημιουργηθεί από τη JavaScript, αλλά εφόσον το απαιτούμενο μέγεθός της είναι γνωστό, οι διαστάσεις της εικόνας μπορούν να χρησιμοποιηθούν για τον υπολογισμό του απαιτούμενου μεγέθους. Συνολικά, το απαιτούμενο μέγεθος του πίνακα μνήμης υπολογίζεται πολλαπλασιάζοντας το ύψος της εικόνας με το πλάτος της επί 4 (επειδή το κάθε εικονοστοιχείο κατέχει μια τιμή για τα κανάλια κόκκινο, μπλε, πράσινο και άλφα, αλλιώς γνωστό ως το μοντέλο χρώματος RGBA). Οι τιμές αυτές αναπαρίστανται στο

memory buffer ως μονοδιάστατος πίνακας ακέραιων αριθμών 8-bit χωρίς πρόσημο. Στην επόμενη γραμμή, καλείται τελικά η WebAssembly επιλέγοντας το επιθυμητό αρχείο .wasm. Αφού γίνει αυτό, οι εξαγωγές των παραδειγμάτων επιστρέφονται στον καλούντα.

Τέλος, η λογική της AssemblyScript φαίνεται στο αρχείο index.ts που βρίσκεται μέσα στον φάκελο assembly. Ο κώδικας είναι ομαδοποιημένος σε εξαγόμενες συναρτήσεις, όπου κάθε συνάρτηση αντιστοιχεί σε έναν συγκεκριμένο χειρισμό εικόνας. Η AssemblyScript ορίζει αυστηρούς τύπους για τις τιμές της, επομένως κάθε παράμετρος και τιμή επιστροφής πρέπει να έχει δηλωμένο τον τύπο της. Ένα κοινό χαρακτηριστικό που μπορεί να παρατηρηθεί μεταξύ όλων των συναρτήσεων είναι ότι μια συγκεκριμένη λογική εφαρμόζεται σε κάθε εικονοστοιχείο μέσω ενός βρόχου for, με κάθε 4 διαδοχικές τιμές που διαβάζονται από τον πίνακα/μνήμη να αντιστοιχούν σε ένα εικονοστοιχείο της εικόνας. Οι εντολές load και store χρησιμοποιούνται για την ανάγνωση και την εγγραφή στο memory buffer. Μέσα στην μνήμη, κάθε στοιχείο έχει μια τιμή μεταξύ 0, που σημαίνει μαύρο (ή απουσία χρώματος), και 255, που σημαίνει τη μέγιστη παρουσία ενός χρώματος. Για παράδειγμα, η τιμή 255 για το κόκκινο ενός εικονοστοιχείου θα σήμαινε ότι το συγκεκριμένο εικονοστοιχείο κατέχει τη μέγιστη ποσότητα κόκκινου χρώματος. Εάν ένα εικονοστοιχείο είναι λευκό, έχει τιμή 255 για όλα τα κόκκινα, μπλε και πράσινα μέσα σε αυτό. Όσον αφορά την 4η τιμή κάθε εικονοστοιχείου, το κανάλι άλφα, δεν χρησιμοποιείται σε καμία από τις συναρτήσεις που υλοποιούνται και επομένως δεν χρησιμοποιείται.

```
export function invert(width: i32, height: i32): void {
  for (let i = 0; i < width * height * 4; i += 4 /*rgba*/) {
    const r = load<u8>(i);
    const g = load<u8>(i + 1);
    const b = load<u8>(i + 2);

    store<u8>(i,      255 - r);
    store<u8>(i + 1, 255 - g);
    store<u8>(i + 2, 255 - b);
  }
}
```

Αν λάβουμε υπόψη τα παραπάνω, είναι εύκολο να κατανοήσουμε την πρώτη συνάρτηση, την `invert`. Αρχικά πρέπει να περαστούν ως παράμετροι το ύψος και το πλάτος της εικόνας επειδή οι τιμές τους θα χρησιμοποιηθούν για την εύρεση των εικονοστοιχείων μέσα στην μνήμη με την χρήση μιάς επανάληψης `for`. Έπειτα οι εντολές `load<u8>()` και `store<u8>()` χρησιμοποιούνται για την ανάγνωση και εγγραφή στην κοινή μνήμη της WebAssembly και της JavaScript. Για να επιτευχθεί λοιπόν η αντιστροφή των χρωμάτων, η συνάρτηση αφαιρεί την τρέχουσα τιμή ενός χρώματος από τη μέγιστη δυνατή τιμή και στη συνέχεια την αποθηκεύει ως νέα τιμή. Έτσι, για παράδειγμα, αν η τιμή ενός χρώματος είναι 25, η ανεστραμμένη τιμή του θα είναι $255-25=230$.

Η λειτουργία που εφαρμόζει φίλτρο κλίμακας του γκρι (grayscale) γίνεται δυνατή με την αφαίρεση των πληροφοριών του χρώματος του κάθε εικονοστοιχείου. Για αυτό χρειάζεται ο υπολογισμός της φωτεινότητας του κάθε εικονοστοιχείου (που ονομάζεται `luma`) ακολουθώντας έναν μαθηματικό τύπο που συνιστάται από τον παγκόσμιο οργανισμό ITU Radiocommunication Sector (ITU-R). Αυτή η τιμή στη συνέχεια χρησιμοποιείται για το κάθε από τα τρία βασικά χρώματα ενός εικονοστοιχείου. Ο μαθηματικός τύπος είναι ο εξής:

$$Y = 0.2126R + 0.7152G + 0.0722B$$

Το `Y` συμβολίζει το `luma`, και τα `R`, `G` και `B` αντιστοιχούν στο κόκκινο, πράσινο, και μπλέ αντίστοιχα.

Η επόμενη συνάρτηση, `basicMonochrome`, λειτουργεί όπως και η προηγούμενη, αλλά με μια πρόσθετη συνθήκη. Υπάρχει τώρα μια οριακή τιμή που αν ξεπεραστεί, το νέο χρώμα θα έχει τη μέγιστη τιμή, διαφορετικά θα έχει την ελάχιστη τιμή. Αυτό οδηγεί σε μια εικόνα με χρωματικό χώρο 2-bit, όπου το φωτεινότερο λευκό και το σκοτεινότερο μαύρο είναι τα μόνα δυνατά χρώματα. Εισάγεται επίσης μια άλλη συνάρτηση που ονομάζεται `randomMonochrome` ως επέκταση της προηγούμενης, όπου το όριο έχει μια τυχαία απόκλιση για κάθε περίπτωση. Αυτό οδηγεί σε μια εικόνα παρόμοια με την προηγούμενη, αλλά τώρα είναι πιο εύκολο να ξεχωρίσει κανείς τα φωτεινά και τα σκοτεινά όρια των αντικειμένων στην εικόνα. Η αλλαγή της τιμής του ορίου και του παράγοντα τυχειότητας μπορεί να οδηγήσει σε διαφορετικά αποτελέσματα. Ο παράγοντας τυχειότητας δίνεται στην `AssemblyScript/WebAssembly` κατά την διαδικασία του `instantiation` της `Wasm`, δίνοντας της ένα `seed`. Η εφαρμογή χρησιμοποιεί την ώρα του υπολογιστή όπως φαίνεται στο παράδειγμα:

```
const wasm = await
WebAssembly.instantiateStreaming(fetch('./build/untouched.wasm')
, {
```

```

env: {
  memory,
  abort: (_msg, _file, line, column) =>
console.error(`Abort at ${line}:${column}`),
  seed: () => new Date().getTime()
}
});

```

Ένας από τους στόχους της εφαρμογής ήταν η υλοποίηση ορισμένων λειτουργιών διόρθωσης χρώματος και για αυτό υλοποιήθηκαν δύο από αυτές: αύξηση/μείωση της φωτεινότητας και του κορεσμού μιας εικόνας. Το χρωματικό μοντέλο RGB επιτρέπει έναν εύκολο τρόπο για την αλλαγή της φωτεινότητας. Για να αυξηθεί ή μειωθεί, οι τιμές του κόκκινου, του μπλε και του πράσινου κάθε εικονοστοιχείου πρέπει να αλλάξουν κατά το ίδιο ποσό. Για να αυξηθεί ο κορεσμός ενός εικονοστοιχείου, πρέπει να αυξηθεί η υψηλότερη τιμή μεταξύ των τιμών του κόκκινου, του μπλε και του πράσινου, καθώς και να μειωθεί η χαμηλότερη τιμή μεταξύ των τριών. Το αντίθετο πρέπει να συμβεί για τη μείωση του κορεσμού. Ένα σημαντικό μειονέκτημα αυτών των συναρτήσεων είναι ότι αποτελούν καταστροφικές λειτουργίες. Αυτό σημαίνει ότι, για παράδειγμα, αν η φωτεινότητα μιας εικόνας αυξηθεί κατά ένα μεγάλο ποσό και στη συνέχεια μειωθεί, τότε αυτό θα καταστρέψει τη συνολική αντίθεση της εικόνας. Αυτό συμβαίνει επειδή δεν έχει υλοποιηθεί κάποιος τρόπος χειρισμού των τιμών που υπερβαίνουν τη μέγιστη ή την ελάχιστη τιμή για τις τιμές RGB.

Η τελευταία λειτουργία που υλοποιήθηκε είναι η οριζόντια αναστροφή (mirroring). Για να επιτευχθεί αυτό, οι τιμές στη μνήμη αυτή την φορά διαβάστηκαν ως ομάδες των 32 bit αντί για 8, γεγονός που επέτρεψε μια ομαλή ανάγνωση και ανταλλαγή τιμών μέσα στον μονοδιάστατο πίνακα της μνήμης.

3.6: Εμπόδια που αντιμετωπίστηκαν κατά την ανάπτυξη

Όπως συμβαίνει και με τα περισσότερα έργα προγραμματισμού, η διαδικασία ανάπτυξης δεν ολοκληρώθηκε χωρίς προβλήματα ή εμπόδια. Η πιο δύσκολη πρόκληση ήταν ίσως η αποσφαλμάτωση του κώδικα AssemblyScript. Οι περισσότερες ώριμες γλώσσες και ιδιαίτερα οι διερμηνευμένες γλώσσες παρέχουν σχεδόν πάντα χρήσιμα μηνύματα σφάλματος και τα συντακτικά σφάλματα μπορούν να βρεθούν και να αντιμετωπιστούν γρήγορα. Στην περίπτωση της AssemblyScript, τα μηνύματα σφάλματος ήταν μερικές φορές αινιγματικά ή μη βοηθητικά, συχνά παρέχοντας μόνο έναν κωδικό σφάλματος και

τίποτε άλλο. Αυτό το ζήτημα ήταν ακόμη πιο δύσκολο να αντιμετωπιστεί όταν επρόκειτο για λογικά σφάλματα, δηλαδή ένα πρόγραμμα που μεταγλωττίζεται με επιτυχία αλλά παράγει σφάλματα κατά την εκτέλεση. Τα σφάλματα σε αυτή την περίπτωση ήταν ακόμη πιο δύσκολο να αντιμετωπιστούν επειδή ο μεταγλωττιστής της AssemblyScript δεν έχει πλέον καμία σχέση με το πρόγραμμα, αλλά το WebAssembly VM του browser έχει αναλάβει να παρέχει ένα χρήσιμο μήνυμα σφάλματος.

Ένα άλλο ζήτημα με την αποσφαλμάτωση είναι ότι η AssemblyScript δεν έχει ενσωματωμένη λειτουργία για την εκτύπωση μηνυμάτων στο τερματικό. Για να γίνει αυτό, πρέπει να εισαχθεί μια συνάρτηση εκτύπωσης από τη JavaScript και να υλοποιηθεί στη AssemblyScript, πράγμα που απαιτεί τη συγγραφή κάποιου κώδικα καθώς και κάποια τεχνογνωσία. Θα ήταν πολύ χρήσιμο να υπήρχε η δυνατότητα χρήσης μιας συνάρτησης `print` κατά τη διάρκεια της ανάπτυξης, αλλά δυστυχώς δεν μπορούσε να υλοποιηθεί με το διαθέσιμο επίπεδο δεξιοτήτων. Καλό θα ήταν να υπήρχε ένα τμήμα γι' αυτό στο `documentation` της AssemblyScript.

Αυτό το θέμα οδηγεί και σε ένα ακόμη ζήτημα: την νεαρή ηλικία της AssemblyScript ως γλώσσα προγραμματισμού. Ενώ πρόκειται για μια εντυπωσιακή τεχνολογία που επιτρέπει στους προγραμματιστές ιστού να δημιουργούν προγράμματα WebAssembly χρησιμοποιώντας ένα οικείο για αυτούς συντακτικό, επί του παρόντος αυτό είναι ταυτόχρονα ένα πλεονέκτημα και μια αδυναμία της γλώσσας, καθώς ορισμένα χαρακτηριστικά που θα μπορούσε ίσως να αναμένεται να λειτουργήσουν τελικά δεν λειτουργούν. Ως παράδειγμα, η συνάρτηση `Math.max()` της JavaScript μπορεί να βρει την υψηλότερη τιμή ένα πλήθος παραμέτρων, αλλά η `Math.max()` της AssemblyScript περιορίζεται στην εύρεση της υψηλότερης τιμής από δύο μόνο παραμέτρους. Ένα παρόμοιο παράδειγμα (που προέρχεται από την αυστηρά τυποποιημένη φύση της AssemblyScript) είναι ότι δεν μπορούν να υπάρξουν δυναμικά αντικείμενα. Κώδικας όπως ο ακόλουθος θα προκαλέσει σφάλμα:

```
let a = {}  
a.prop = "hello world"
```

Τα αντικείμενα πρέπει να είναι τυποποιημένα, όπως είναι ένα `Map` ή μία κλάση:

```
// Map  
let a = new Map<string,string>()  
a.set("prop", "hello world")
```

```
// class
class A {
  constructor(public prop: string) {}
}
var a = new A("hello world")
```

Μερικές από αυτές τις ιδιορρυθμίες δεν προέρχονται καν από την ίδια την AssemblyScript αλλά από την WebAssembly που δεν έχει υλοποιήσει συγκεκριμένα απαραίτητα χαρακτηριστικά για την υποστήριξη των αναμενόμενων λειτουργιών. Ευτυχώς, ακόμη και αν μια τέτοια συμπεριφορά μπορεί να είναι ενοχλητική, το documentation της AssemblyScript είναι αρκετά λεπτομερές στην τεκμηρίωση τυχόν ελλείψεων ή μη αναμενόμενων συμπεριφορών. Επίσης, η AssemblyScript είναι μια γλώσσα που αναπτύσσεται από μια κοινότητα προγραμματιστών και δεν έχει την υποστήριξη ενός μεγάλου ιδρύματος (όπως η Rust, που πλέον ανήκει στο ίδρυμα Linux), αλλά υπάρχει αισιοδοξία ότι σιγά σιγά θα καταφέρει να εξομαλύνει τις περισσότερες ιδιορρυθμίες της.

Ένα τελευταίο ζήτημα που συναντήθηκε κατά τη διάρκεια της ανάπτυξης ήταν η εργασία πάνω στην κοινή μνήμη μεταξύ JavaScript και AssemblyScript. Αναπαρίσταται ως ένα ArrayBuffer με δυνατότητα αλλαγής μεγέθους, δηλαδή έναν πίνακα από bytes. Τόσο η WebAssembly όσο και η JavaScript μπορούν να δημιουργήσουν τέτοια αντικείμενα μνήμης και αμφότερα μπορούν να εξάγουν/εισάγουν αυτά τα αντικείμενα μεταξύ τους. Ωστόσο, όλα αυτά είναι πιο εύκολο να ειπωθούν παρά να γίνουν, λόγω των πολλών σφαλμάτων μνήμης που μπορούν να προκύψουν αν ο προγραμματιστής δεν κατανοεί απόλυτα τις ενέργειες του. Ως προσωπική άποψη, ο προγραμματιστής θα πρέπει να κατανοήσει πρώτα σε βάθος τον τρόπο με τον οποίο μπορεί να δημιουργηθεί και να χρησιμοποιηθεί η μνήμη της WebAssembly πριν την χρήση της στον κώδικα από τον ίδιο.

3.7: Διαδικασία εγκατάστασης και χρήσης

Η εφαρμογή φιλοξενείται στο διαδίκτυο (με το GitHub Pages) και είναι διαθέσιμη για δημόσια χρήση, ενώ ο ίδιος ο κώδικας είναι επίσης open source και φιλοξενείται στο GitHub. Για να γίνουν τοπικές δοκιμές, αλλαγές, ή ρυθμίσεις, πρέπει πρώτα να γίνει ένας "κλώνος" git με την εντολή git clone:

```
git clone https://github.com/danielyor/image-manip.git
```

Το αρχείο README.md προσφέρει κάποιες οδηγίες για την εγκατάσταση, οι οποίες είναι οι εξής:

1. Εγκατάσταση εξαρτήσεων με την εντολή: `npm i`
2. Εκτέλεση του script κατασκευής με την εντολή: `npm run asbuild`
3. Φιλοξενία ενός τοπικού διακομιστή ιστού για προβολή με την εντολή: `npx ws -p 1234`

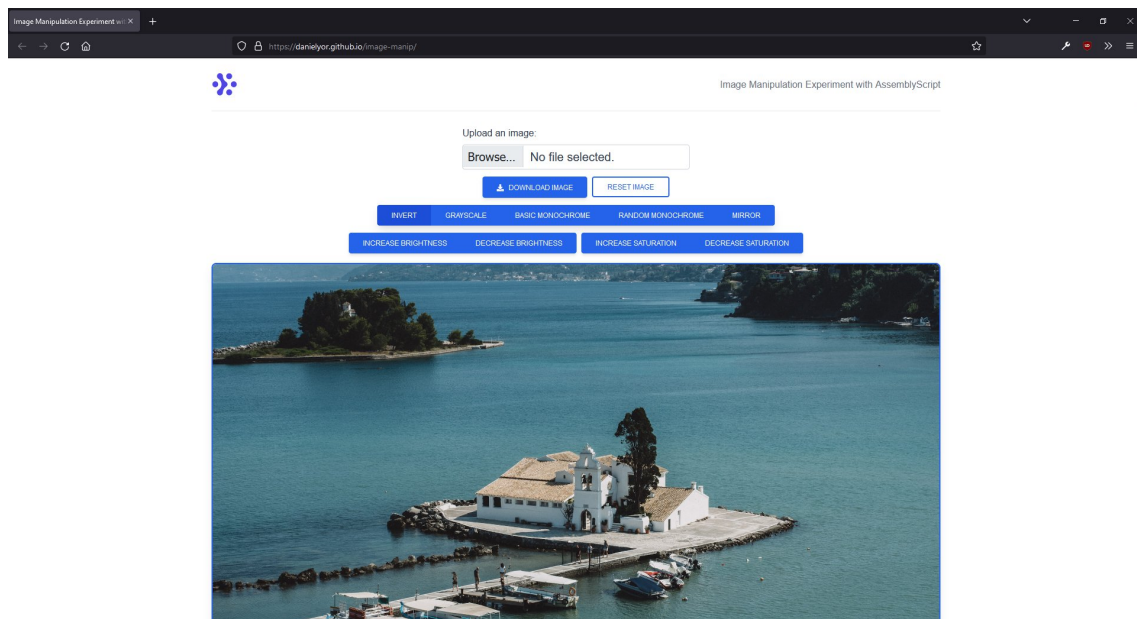
Η εφαρμογή είναι τώρα διαθέσιμη για προβολή και χρήση στη διεύθυνση <http://localhost:1234>. Δεδομένου ότι η εφαρμογή φορτώνει το αρχείο WebAssembly (.wasm) και όχι το αρχείο AssemblyScript (.ts), μετά από κάθε επεξεργασία στον κώδικα AssemblyScript η εντολή `npm run asbuild` πρέπει να ενεργοποιηθεί ξανά. Όσον αφορά το ποιο αρχείο .wasm φορτώνεται (βελτιστοποιημένο ή μη βελτιστοποιημένο), αυτό μπορεί να επιλεγεί μέσα από το αρχείο index.js, στην συνάρτηση φόρτωσης WebAssembly του κώδικα.

Από την άποψη του χρήστη, η εφαρμογή είναι απλή στη χρήση. Οι λειτουργίες είναι τοποθετημένες στο επάνω μέρος της σελίδας ως κουμπιά και ομαδοποιημένες ανά είδος (διόρθωση εικόνας, φίλτρο κλπ.). Μια προεπιλεγμένη εικόνα φορτώνεται για σκοπούς επίδειξης, αλλά μπορεί επίσης να φορτωθεί οποιαδήποτε άλλη εικόνα που μπορεί να θέλει ο χρήστης, με υποστήριξη των πιο κοινών μορφών εικόνας όπως .jpg και .png. Λόγω της υπολογιστικής λογικής πίσω από τις λειτουργίες και ανάλογα με το μέγεθος της εικόνας, ορισμένες λειτουργίες ενδέχεται να χρειάζονται περισσότερο χρόνο για να ολοκληρωθούν από άλλες. Οι εικόνες Full HD έχουν αποδειχθεί (από εμπειρικές δοκιμές) ότι λειτουργούν πολύ καλά σε έναν desktop υπολογιστή. Η διεπαφή χρήστη για κινητά παρέχει μια εξίσου ικανοποιητική εμπειρία.

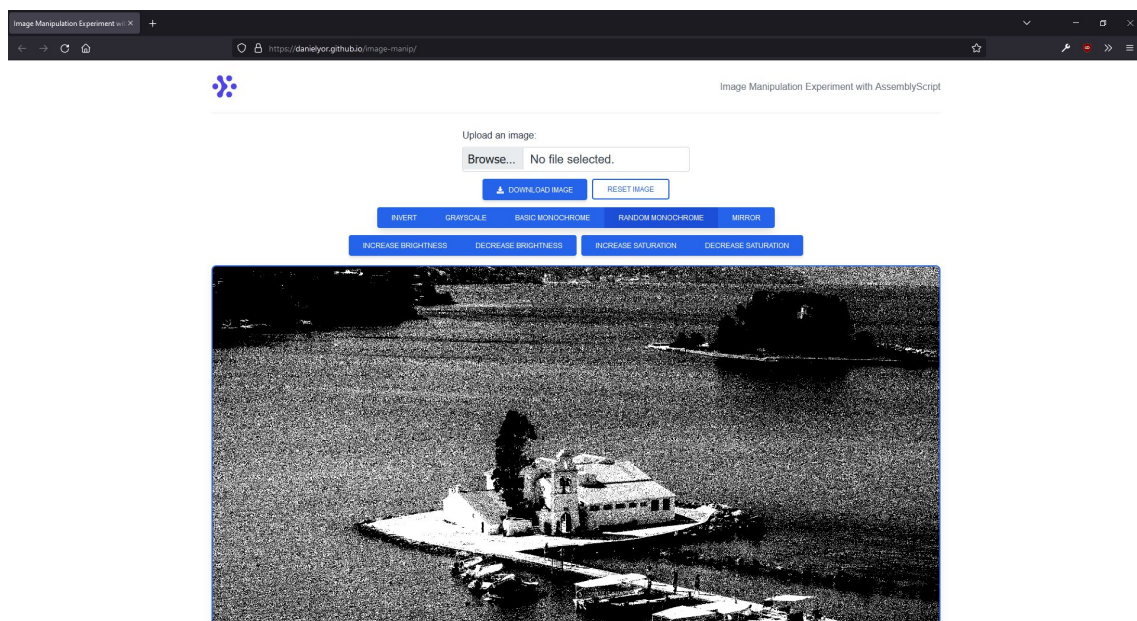
3.8: Αποτελέσματα και δυνατότητες επέκτασης

Συμπερασματικά, η AssemblyScript παρέχει ένα αξιοπρεπές σημείο εισόδου στην WebAssembly για προγραμματιστές με υπόβαθρο JavaScript ή TypeScript, προσφέροντας οικείο συντακτικό ή/και οικοσύστημα. Πρόκειται για ένα εντυπωσιακό κομμάτι τεχνολογίας αλλά με κάποιες επιφυλάξεις, επειδή αν ένας προγραμματιστής θα προσπαθήσει να την χρησιμοποιήσει, θα πρέπει πρώτα να βεβαιωθεί ότι έχει εξοικειωθεί σε βάθος με τις βασικές έννοιες της WebAssembly και τον τρόπο χρήσης τους, καθώς και με το documentation της AssemblyScript. Η ίδια η WebAssembly έχει αποδειχθεί πολύ κατάλληλη για εργασίες όπως η βαριά επεξεργασία εικόνας και υπάρχει ελπίδα ότι περισσότερα έργα και προγραμματιστές θα μπορέσουν σύντομα να δουν τις δυνατότητες της και να τη βοηθήσουν να εξελιχθεί σε ένα πραγματικό πρότυπο του ιστού.

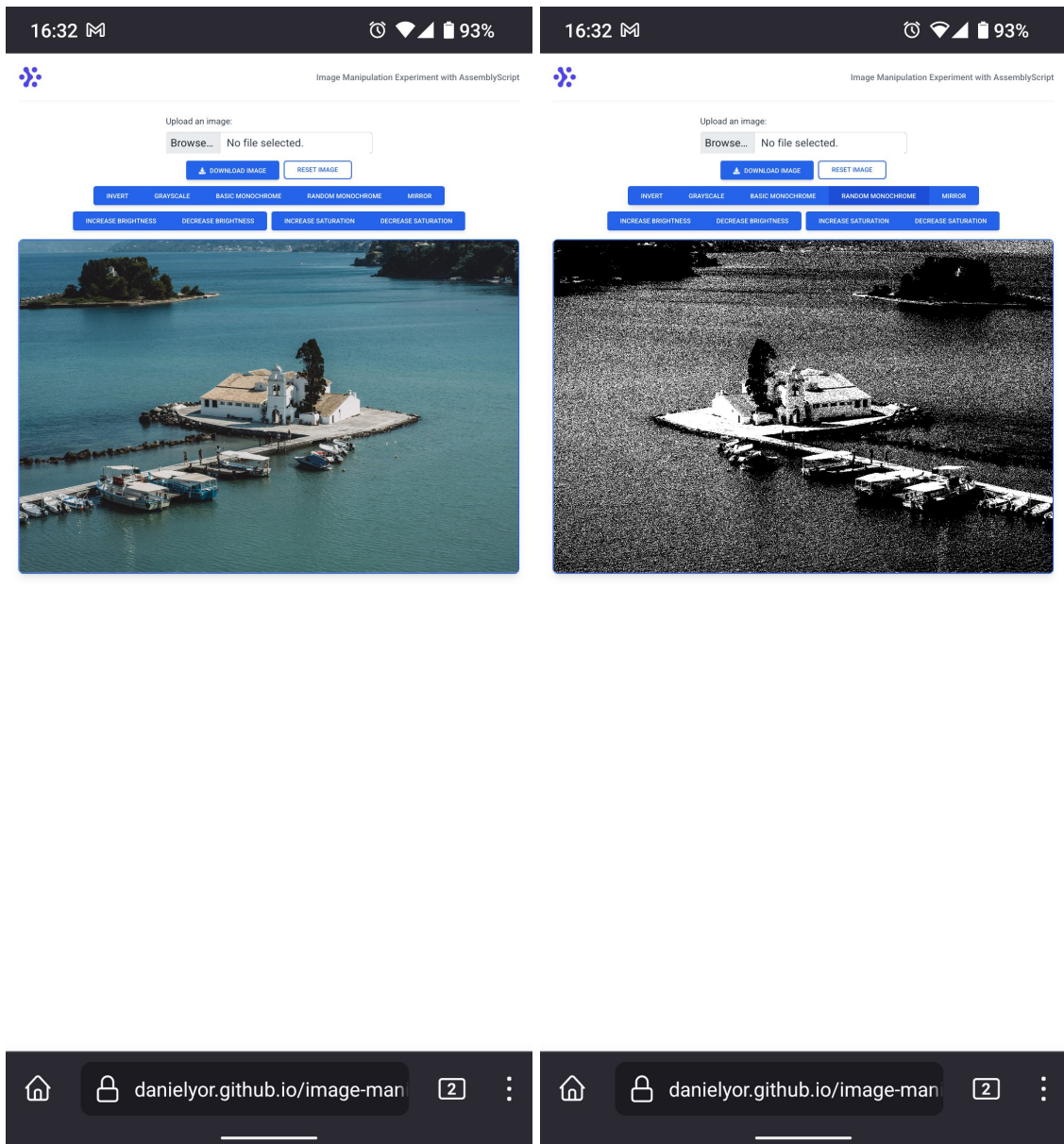
Και πρίν κλείσει και αυτό το κεφάλαιο, θα προταθούν κάποιες προτάσεις για τη βελτίωση της ίδιας της εφαρμογής. Μια ιδέα είναι ότι, δεδομένου ότι η WebAssembly είναι φορητή και μπορεί να λειτουργήσει εκτός του ιστού, η εφαρμογή θα μπορούσε να μεταφερθεί και σε άλλα περιβάλλοντα, όπως είναι μια εφαρμογή για υπολογιστές ή smartphone, ή ίσως ακόμη και μια εφαρμογή τερματικού. Σίγουρα θα μπορούσαν να προστεθούν νέα χαρακτηριστικά από κάποιον με την κατάλληλη τεχνογνωσία, και ορισμένα από τα τρέχοντα θα μπορούσαν σίγουρα να βελτιωθούν με την εφαρμογή πιο αποτελεσματικών αλγορίθμων και με τη βελτιστοποίηση του κώδικα. Τέλος, το ίδιο το UI θα μπορούσε να βελτιωθεί με καλύτερα και πιο λεπτομερή στοιχεία ελέγχου, πχ. με ράβδους ολίσθησης ή πεδία αριθμών για την επιλογή της έντασης των χειρισμών, ή την προσθήκη κάποιας ανατροφοδότησης που να υποδεικνύει μια λειτουργία σε εξέλιξη.



Στιγμιότυπο 1: Η αρχική σελίδα της εφαρμογής σε browser Firefox



Στιγμιότυπο 2: Η αρχική εικόνα μετά τις επεξεργασίες Mirror και Random Monochrome



Στιγμιότυπα 3 και 4: Οι προηγούμενες ενέργειες σε smartphone Android στην εφαρμογή Firefox Mobile

4ο Κεφάλαιο:

Συμπεράσματα και Μελλοντικές Βλέψεις

Με την ολοκλήρωση της παρούσας πτυχιακής εργασίας, μπορούν να διατυπωθούν ορισμένα συμπεράσματα.

Αρχικά, η εισαγωγή της WebAssembly επιλύει πραγματικά αναδυόμενα προβλήματα και νέες ανάγκες που σχετίζονται με τη διαδικτυακή πλατφόρμα, όπως για παράδειγμα η ανάγκη για την δημιουργία διαδικτυακών εφαρμογών υψηλών αποδόσεων. Ταυτόχρονα, επιτρέπει μεγαλύτερη ελευθερία επιλογών για τους προγραμματιστές ιστού, καθώς έχουν πλέον στη διάθεσή τους περισσότερα εργαλεία με διαφορετικές δυνατότητες (εκτός της JavaScript και των frameworks της) για τη δημιουργία frontend εφαρμογών ιστού. Και επίσης, αν ληφθεί υπόψη η ταχύτητα ανάπτυξης της WebAssembly, η τυποποίηση και η υποστήριξη της από αρκετούς οργανισμούς και όλους τους browsers με το μεγαλύτερο μερίδιο της αγοράς, αυτό οδηγεί στο συμπέρασμα ότι η WebAssembly είναι μια επιτυχής τεχνολογία εν εξελίξει.

Όταν εφαρμόζεται στον τομέα του χειρισμού εικόνων, συμπεραίνεται ότι η WebAssembly ταιριάζει εξαιρετικά στις διαδικτυακές εφαρμογές επεξεργασίας εικόνας, καθώς μπορεί να επιτρέψει τη δυνατότητα δημιουργίας εφαρμογών που μπορούν να εκτελούν πιο υπολογιστικά απαιτητικές λειτουργίες ή/και να είναι διαθέσιμες και σε άλλες πλατφόρμες, όπως τα smartphones ή άλλα λειτουργικά συστήματα. Ωστόσο, υπάρχουν και κάποια μειονεκτήματα: δεδομένου ότι το οικοσύστημα της WebAssembly δεν διαθέτει την ωριμότητα που απολαμβάνει η JavaScript, η διαδικασία ανάπτυξης μπορεί να είναι ενδεχομένως πιο πολύπλοκη και αχαρτογράφητη, αν και αυτό αναμένεται να βελτιωθεί.

Βιβλιογραφία - Δικτυογραφία

1. [Δελτίο τύπου ανακοίνωσης της JavaScript](#), PR Newswire, 4 Δεκεμβρίου 1995
2. ["Ajax"](#), MDN Web Docs, ανακτημένο 20 Αυγούστου 2022
3. ["World Wide Web Timeline"](#), Pew Research Center, 11 Μαρτίου 2014
4. [WebAssembly Specification](#), webassembly.github.io, ανακτημένο 20 Αυγούστου 2022
5. ["WebAssembly"](#), MDN Web Docs, ανακτημένο 20 Αυγούστου 2022
6. [TypeScript Documentation](#), typescriptlang.org, ανακτημένο 20 Αυγούστου 2022
7. ["Luma \(video\)"](#), Wikipedia, ανακτημένο 27 Αυγούστου 2022
8. [AssemblyScript](#), The AssemblyScript Project, ανακτημένο 20 Αυγούστου 2022
9. ["The AssemblyScript Book"](#), The AssemblyScript Project, ανακτημένο 20 Αυγούστου 2022
10. [Empscripten](#), GitHub, ανακτημένο 27 Αυγούστου
11. [Cheerp](#), GitHub, ανακτημένο 27 Αυγούστου
12. ["WebAssembly - Rust Programming Language"](#), GitHub, ανακτημένο 27 Αυγούστου
13. [Blazor](#), GitHub, ανακτημένο 27 Αυγούστου
14. [TeaVM](#), GitHub, ανακτημένο 27 Αυγούστου
15. [JWebAssembly](#), GitHub, ανακτημένο 27 Αυγούστου
16. [Pyodide](#), GitHub, ανακτημένο 27 Αυγούστου
17. [Wasmer](#), GitHub, ανακτημένο 27 Αυγούστου
18. [Wasmtime](#), GitHub, ανακτημένο 27 Αυγούστου
19. [WasmEdge](#), GitHub, ανακτημένο 27 Αυγούστου
20. [WebAssembly Micro Runtime](#), GitHub, ανακτημένο 27 Αυγούστου