ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΥΓΚΡΙΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΤΕΧΝΙΚΩΝ ΑΠΟΘΗΚΕΥΣΗΣ ΔΕΔΟΔΕΜΕΝΩΝ ΣΤΗΝ ΠΛΕΥΡΑ ΤΟΥ ΠΕΛΑΤΗ (CLIENT-SIDE DATA STORAGE).

Διπλωματική Εργασία

του

Αναστάσιου Καλούδη

Θεσσαλονίκη, 11 / 2022

ΣΥΓΚΡΙΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΤΕΧΝΙΚΩΝ ΑΠΟΘΗΚΕΥΣΗΣ ΔΕΔΟΔΕΜΕΝΩΝ ΣΤΗΝ ΠΛΕΥΡΑ ΤΟΥ ΠΕΛΑΤΗ (CLIENT-SIDE DATA STORAGE).

Αναστάσιος Καλούδης

Μηχανικός Πληροφορικής 2013, Αλεξάνδρειο Τεχνολογικό Ιδρυμα Θεσσαλονίκης

Διπλωματική εργασία

Υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΈΝΗ ΠΛΗΡΟΦΟΡΙΚΉ

Επιβλέπων Καθηγητής

Θεόδωρος Κασκάλης

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την …/…/……

Κασκάλης Θεόδωρος	Γεωργιάδης Χρήστος	Κολωνιάρη Γεωργία
.....	.....	.....

Αναστάσιος Καλούδης

Περίληψη

Οι σύγχρονοι φυλλομετρητές ιστού υποστηρίζουν διάφορους τρόπους αποθήκευσης δεδομένων από ιστότοπους στον υπολογιστή του χρήστη και στη συνέχεια τα ανακτούν όταν είναι απαραίτητο. Αυτό επιτρέπει την αποθήκευση και την διαχείριση δεδομένων για μακροπρόθεσμη αποθήκευση. Έτσι μπορούμε να αποθηκεύσουμε ιστότοπους ή έγγραφα για χρήση εκτός σύνδεσης, να διατηρούνται ρυθμίσεις για τον ιστότοπό για συγκεκριμένους χρήστες και πολλά άλλα. Οι περισσότεροι ιστότοποι είναι δυναμικοί και αποθηκεύουν δεδομένα από το διακομιστή χρησιμοποιώντας κάποιο είδος βάσης δεδομένων (αποθήκευση από την πλευρά του διακομιστή), στη συνέχεια εκτελούν κώδικα από την πλευρά του διακομιστή για να ανακτήσουν τα απαραίτητα δεδομένα, να τα εισαγάγουν σε πρότυπα στατικών σελίδων και να εξυπηρετήσουν το HTML που προκύπτει στη πλευρά του πελάτη που εμφανίζεται από το πρόγραμμα περιήγησης του χρήστη. Συχνά ο χώρος αποθήκευσης από την πλευρά του πελάτη και του διακομιστή χρησιμοποιούνται μαζί. Γνωστές μέθοδοι είναι η χρήση βάσης δεδομένων, αποθήκευση σε τοπικά αρχεία, χρήση τοπικών session κ.α. Για τις ανάγκες της παρούσας εργασίας δημιουργήθηκε μια εφαρμογή ενός ιστότοπου όπου μετρήθηκαν οι χρόνοι ανάκτησης δεδομένων με χρήση του LocalStorage, του IndexedDB και του WebSQL. Ο μικρότερος χρόνος ανάκτησης για τα ίδια δεδομένα παρατηρήθηκε στην τεχνική LocalStorage, μετά ακολούθησε η τεχνική WebSQL με αρκετά μεγάλη διαφορά ενώ με μικρότερη διαφορά ακολουθούσε η τεχνική IndexedDB.

Λέξεις Κλειδιά: Αποθήκευση Δεδομένων στην Πλευρά του Πελάτη, LocalStorage, IndexedDB, WebSQL

Abstract

Modern web browsers support various ways of storing data from websites on the user's computer and then retrieve it when needed. This allows data to be stored and managed for long-term storage. This allows us to save websites or documents for offline use, maintain site settings for specific users, and more. Most sites are dynamic and store data from the server using some kind of database (server-side storage), then execute server-side code to retrieve the necessary data, import it into static page templates, and serve the resulting HTML client to be displayed by the user's browser. Often the client and server storage are used together. Known methods are the use of database, storage in local files, use of local sessions etc. For the purposes of this work, a web application was created where data retrieval times were measured using LocalStorage, IndexedDB, and WebSQL. The shortest recovery time for the same data was observed in the LocalStorage technique, followed by the WebSQL technique with a large difference while with a smaller difference the IndexedDB technique followed.

Keywords: Client Side Data Storage, LocalStorage, IndexedDB, WebSQL

Περιεχόμενα

1. Εισαγωγή

Τα τελευταία χρόνια, από τότε που ο οργανισμός W3C εισήγαγε τις προδιαγραφές HTML5, έχει γίνει μεγάλη προσπάθεια για την εφαρμογή νέων τεχνολογιών αποθήκευσης από την πλευρά του πελάτη ούτως ώστε να δωθεί στους προγραμματιστές η ευκαιρία να δημιουργήσουν ισχυρές εφαρμογές Ιστού που χρησιμοποιούν HTML5 και CSS3. Οι προγενέστερες τεχνολογίες αποθήκευσης από την πλευρά του πελάτη, όπως τα cookies HTTP ή οι παράμετροι URL, δεν ήταν αρκετές λόγου περιορισμένου μεγέθους δεδομένων και ασφάλειας. Η έλλειψη εξελιγμένων τεχνολογιών αποθήκευσης από την μεριά του πελάτη (client-side storage) σε συνδυασμό με την αλματώδη ανάπτυξη και χρήση εφαρμογών για κινητές συσκευές, οδήγησε στην επέκταση των thick(βαριών) εγγενών εφαρμογών πελατών-υπολογιστών. Μια τέτοια εφαρμογή που χρησιμοποιεί την αρχιτεκτονική πελάτη-διακομιστή δεν είναι φορητή και πρέπει να υλοποιηθεί για κάθε πλατφόρμα κινητής τηλεφωνίας (Android, iOS, BlackBerry, Windows Phone κ.λπ.). Τα τελευταία χρόνια υπάρχουν όλο και περισσότεροι χρήστες του διαδικτύου με ολοένα και περισσότερες απαιτήσεις σε πόρους συστημάτων, αυτό σε συνάρτηση με την περιορισμένη αποτελεσματικότητα των διακομιστών δημιούργησε την ανάγκη για τον επιμερισμό των πόρων ανάμεσα σε πελάτη και διακομιστή. Η αποθήκευση δεδομένων στην πλευρά του πελάτη είναι ένας προφανής τρόπος αύξησης της απόδοσης των εφαρμογών και ελάφρυνση των πόρων του διακομιστή (ειδικά τώρα που οι υπολογιστές-πελάτες έχουν σχετικά μεγάλες δυνατότητες CPU και RAM).

Οι νέες τεχνολογίες αποθήκευσης από την πλευρά του πελάτη θα μπορούσαν να χρησιμοποιηθούν και σε περιπτώσεις όπου υπάρχει πρόβλημα με την σύνδεση του διαδικτύου (Offline). Ειδικότερα μπορούν να βελτιωθούν τα κατανεμημένα συστημάτα απόκτησης δεδομένων σε διάφορους τομείς.

Στην εργασία τους οι (Janik, & Kiebzak, 2014) συνόψισαν τα αποτελέσματα της έρευνας και της μελέτης βιωσιμότητας των παρακάτω τριών τεχνολογιών: του χώρου αποθήκευσης Ιστού (Web Storage), της ευρετηριασμένης βάσης δεδομένων(Indexed Database) και της API αρχείων(File Api) οι οποίες είναι σχετικά σύγχρονες τεχνολογίες και εξακολουθούν να χρειάζονται ενδελεχή εξέταση. Στο πρώτο μέρος της εργασίας παρουσιάστηκαν προηγούμενοι τρόποι αποθήκευσης δεδομένων σε ένα πρόγραμμα περιήγησης ιστού από την πλευρά του πελάτη. Εν συνεχεία, συνοψίζεται

ο χώρος αποθήκευσης Ιστού, η βάση δεδομένων ευρετηρίου, η API αρχείων. Παρουσιάστηκε επίσης το σχέδιο ενός framework και η υλοποίηση αναφοράς μιας βιβλιοθήκης για γλώσσα Java που μπορεί να χρησιμοποιηθεί για την ενοποίηση διαδικτυακών εφαρμογών γραμμένων σε Java (web frames που βασίζονται στην τεχνολογία JSP) με τις αναφερόμενες τεχνολογίες πελάτη.

2. Ιστορική εξέλιξη στην αποθήκευση δεδομένων στην πλευρά του πελάτη

2.1. Cookies HTTP

Τα HTTP Cookies είναι επίσης γνωστά και σαν web cookies, Internet cookies, browser cookies και εμφανίστηκαν το 1994. Ο βασικότερος λόγος δημιουργίας τους ήταν η δυνατότητα αποθήκευσης της κατάστασης του καλαθιού των πελατών σε διαδικτυακά καταστήματα.

Τα cookies δημιουργούνται από τον διακομιστή καθώς ένα χρήστης περιηγείται σε έναν ιστότοπο και στην συνέχεια αποθηκεύονται ως αρχεία στο σύστημα αρχείων ενός πελάτη. Προφανώς το πρόγραμμα περιήγησης ιστού μπορεί να έχει πρόσβαση σε αυτά τα αρχεία. Το κάθε cookie αποτελεί μια μικρή πληροφορία που αποτελείται από μοναδικό κλειδί και την αξία του. Όταν ο διακομιστής HTTP θέλει να ορίσει ένα cookie σε έναν συγκεκριμένο υπολογιστή-πελάτη, στέλνει ένα κλειδί και μια τιμή του cookie στην απόκριση HTTP.

Στη συνέχεια, το πρόγραμμα περιήγησης τα στέλνει πίσω στον διακομιστή με κάθε αίτημα HTTP, εισάγοντας καταστάσεις (μνήμη προηγούμενων συμβάντων) σε (stateless) HTTP συναλλαγές. επισυνάπτει όλα τα προηγουμένως αποθηκευμένα cookies σε μια κεφαλίδα(header) στο αίτημα HTTP, επομένως όλα τα cookies για έναν σχετικό web domain αποστέλλονται στον διακομιστή. Τα μειονεκτήματα της χρήσης του αιτήματος HTTP για τη μεταφορά cookie είναι ο περιορισμός του μεγέθους των cookies (4 kB) , ο περιορισμός του αριθμού (50 για κάθε domain) και ότι ο browser μπορεί να υποστηρίξει μέχρι 3000 cookies στον αριθμό.

Μια άλλη αδυναμία των cookies είναι ότι η τεχνολογία είναι επιρρεπής σε επιθέσεις Ιστού. Παρόλο που έχουν εισαχθεί ορισμένοι μηχανισμοί για την πρόληψη τέτοιων απειλών, ωστόσο δεν είναι δυνατό να αντιμετωπισθούν όλες τις επιθέσεις (West & Pulimood, 2012). Σήμερα, τα cookies HTTP εξακολουθούν να χρησιμοποιούνται

ευρέως από εφαρμογές Ιστού και σε ιστοσελίδες, κυρίως για την αποθήκευση του αναγνωριστικού περιόδου σύνδεσης αλλά και για την εξατομίκευση του ιστότοπου. Τέλος τα cookies μπορούν να χρησιμοποιηθούν για την παρακολούθηση των δραστηριοτήτων των χρηστών σε ιστότοπους.

2.2. Παράμετροι URL – Query Strings

Σε ορισμένες περιπτώσεις, τα cookies μπορούν να αντικατασταθούν από παραμέτρους URL γεγονός πολύ σημαντικό αφού οι χρήστες μπορούν να απενεργοποιήσουν τα cookies σε προγράμματα περιήγησης ιστού. Επομένως, οι παράμετροι URL χρησιμοποιούνται συχνά για τη διατήρηση της περιόδου σύνδεσης των χρηστών. Η κύρια διαφορά μεταξύ αυτού του μηχανισμού και των cookies είναι το γεγονός ότι οι πληροφορίες που μεταφέρονται σε παραμέτρους URL χάνονται μόλις ο χρήστης κλείσει ένα παράθυρο του προγράμματος περιήγησης Ιστού. Τα cookies διαρκούν στο σύστημα αρχείων των χρηστών. Βασικό μειονέκτημα τους είναι ότι το μέγεθος τους περιορίζεται στο επιτρεπτό μήκος του URL.

2.3. HTML5 Manifest

Η προδιαγραφή HTML5 περιέχει ένα αξιοσημείωτο χαρακτηριστικό, το οποίο είναι ένα αρχείο δήλωσης (μανιφέστο). Στο αρχείο του οποίου η τοποθεσία αναφέρεται σε έγγραφα HTML εφαρμογών Ιστού, οι προγραμματιστές μπορούν να καθορίσουν τις σελίδες που πρέπει να αποθηκευτούν προσωρινά σε προγράμματα περιήγησης Ιστού. Αυτή η δυνατότητα μπορεί να χρησιμοποιηθεί για την παροχή μιας διαδικτυακής εφαρμογής εκτός σύνδεσης. Ένα αρχείο δήλωσης είναι μια εξαιρετική ιδέα, καθώς επιτρέπει σε έναν προγραμματιστή να επηρεάζει ενεργά τη συμπεριφορά εφαρμογών μερικώς εκτός σύνδεσης.

2.4. Αποθήκευση Ιστού

Το Web Storage είναι ένα σύνολο ζευγών κλειδιών-τιμών που είναι αποθηκευμένα σε ένα πρόγραμμα περιήγησης Ιστού για κάθε προέλευση(για κάθε domain και πρωτόκολο)[1]. Χωρίζεται στην Τοπική αποθήκευση (που είναι μόνιμη) και στην Αποθήκευση περιόδου λειτουργίας (όπου αποθηκεύονται δεδομένα για την διάρκεια ζωής ενός παραθύρου προγράμματος περιήγησης Ιστού). Μια εφαρμογή Ιστού μπορεί να αποθηκεύσει μόνο συμβολοσειρές, το οποίο σημαίνει ότι για να αποθηκευτεί το αντικείμενο JavaScript, πρέπει να γίνει σειριακή (μετατροπή σε μορφή JSON) πριν. Η

[1] http://www.w3.org/TR/webstorage

μετατροπή αντικειμένων σε JSON μπορεί να είναι χρονοβόρα, επομένως οι προγραμματιστές θα πρέπει να το γνωρίζουν. Είναι ιδιαίτερα σημαντικό επειδή οι κλήσεις στο WebStorage API είναι σύγχρονες. Μια ακόμα σημαντική διαφορά σε σχέση με τα cookies είναι ότι το όριο χρήσης του μεγέθους τους μπορεί να φτάσει τα 5MB.

2.5. IndexedDB

Η IndexedDB είναι μια αντικειμενοστραφής βάση δεδομένων που βασίζεται σε javascript, η οποία είναι ενσωματωμένη σε ένα πρόγραμμα περιήγησης ιστού[2]. Μπορεί να αποθηκεύσει περισσότερους τύπους δεδομένων από το Web storage api, επειδή είναι σε θέση να χειριστεί κάθε αντικείμενο που υποστηρίζεται από τον αλγόριθμο «δομημένου κλώνου». Ένα domain μπορεί να έχει πολλές βάσεις δεδομένων (που προσδιορίζονται με ένα όνομα και μια έκδοση) και η καθεμία μπορεί να έχει πολλούς χώρους αποθήκευσης δεδομένων, έτσι ώστε ένας χώρος αποθήκευσης να έχει τον δικό του ορισμό ενός κλειδιού και να έχει σχεδιαστεί για να αποθηκεύει παρόμοια αντικείμενα.

Ομοίως με το WebStorage, τα δεδομένα στο χώρο αποθήκευσης του IndexedDB αντιπροσωπεύονται από ένα ζεύγος ενός κλειδιού και μιας τιμής. Το κλειδί μπορεί να δημιουργηθεί αυτόματα ή να οριστεί μέσω προγραμματισμού (στην πραγματικότητα, το κλειδί είναι επίσης ένα από τα χαρακτηριστικά των αποθηκευμένων αντικειμένων). Προφανώς, τα κλειδιά πρέπει να είναι μοναδικά σε κάθε χώρο αποθήκευσης.

Το IndexedDB έχει δύο API – το ασύγχρονο και το σύγχρονο. Συνιστάται η χρήση του ασύγχρονου API από τον κώδικα JavaScript της εφαρμογής web, καθώς μπορεί να λειτουργήσει σε αρκετά μεγάλα σύνολα δεδομένων. Η εκτέλεση κώδικα JavaScript μπορεί να "κρεμάσει" μια καρτέλα προγράμματος περιήγησης ιστού ή ακόμα και ολόκληρο το παράθυρο (ανάλογα με τον τρόπο υλοποίησης ενός δεδομένου προγράμματος περιήγησης ιστού).

Το εξαιρετικό χαρακτηριστικό του IndexedDB, είναι ότι επιτρέπει στον προγραμματιστή να ορίσει ευρετήρια για επιλεγμένα χαρακτηριστικά για ένα επιλεγμένο χώρο αποθήκευσης δεδομένων . Όπως οι canonical βάσεις δεδομένων έτσι και η IndexedDB , βελτιώνει την απόδοση κατά την αναζήτηση οντοτήτων.

[2] http://www.w3.org/TR/IndexedDB

Επιπλέον, τα ευρετήρια μπορούν να χρησιμοποιηθούν ως περιορισμοί, προστατεύοντας έτσι τη μοναδικότητα των χαρακτηριστικών.

Η IndexedDB είναι συναλλακτική. Κάθε λειτουργία στα δεδομένα της εκτελείται σε ένα πεδίο συναλλαγής. Ο εκάστοτε προγραμματιστής μπορεί να επιλέξει μεταξύ δύο τύπων συναλλαγής – μόνο για ανάγνωση και ανάγνωσης-εγγραφής – ανάλογα με τον τύπο χειρισμού δεδομένων που απαιτείται. Υπάρχει επίσης ένας άλλος τύπος συναλλαγής - "αλλαγή έκδοσης", ο οποίος όμως χρησιμοποιείται μόνο κατά τη δημιουργία μιας βάσης δεδομένων ή για την αλλαγή της έκδοσής της.

Αξίζει να αναφερθεί ότι υπήρξαν ορισμένες ιδέες συγχρονισμού δεδομένων από διακομιστή (με αναπτυγμένη βάση δεδομένων SQL) με πελάτη web που εκτελεί διαφορετικούς μηχανισμούς (IndexedDB, WebSQL Database, Google Gears). Οι συγγραφείς της μελέτης (Janik & Kiebzak, 2014) αποφάσισαν να δημιουργήσουν έναν προσαρμοσμένο διερμηνέα SQL (Leblon,2010).

Μια ενδιαφέρουσα περίληψη σχετικά με τη χρήση των παραπάνω τεχνολογιών JavaScript μπορεί να βρεθεί στην εργασία του (Laine, 2012).

2.6. API αρχείου

Το W3C εισήγαγε το File API για το χειρισμό αρχείων σε ένα πρόγραμμα περιήγησης ιστού πριν αυτά σταλούν σε έναν διακομιστή ιστού[3]. Αυτή η προσέγγιση επιτρέπει την εκτέλεση ορισμένων λειτουργιών σε αρχεία χρησιμοποιώντας ισχύ CPU του πελάτη (εξοικονομώντας έτσι πόρους του διακομιστή). Επιπλέον, τα αρχεία μπορούν να διαβαστούν (ως δυαδική συμβολοσειρά, κείμενο ή DataURL) και το περιεχόμενό τους μπορεί να αποθηκευτεί στο Web Storage ή στο IndexedDB.

Μια άλλη εξαιρετική λειτουργία του File API είναι ένα σύστημα αρχείων sandbox[4], στο οποίο η εφαρμογή Ιστού μπορεί να έχει πρόσβαση για να λειτουργεί[5] σε αρχεία (δημιουργία, εγγραφή, ανάγνωση και διαγραφή) και φακέλους. Υπάρχουν δύο τύποι συστημάτων αρχείων – τα μόνιμα και τα προσωρινά. Κατά τη χρήση του προσωρινού συστήματος αρχείων, ένα πρόγραμμα περιήγησης ιστού φροντίζει για τη διάρκεια ζωής των αρχείων – εάν ένα πρόγραμμα περιήγησης ιστού χρειάζεται λίγο χώρο στο δίσκο, μπορεί να διαγράψει αρχεία χωρίς προειδοποίηση. Αλλά όταν μια εφαρμογή

[3] http://www.w3.org/TR/FileAPI
[4] http://www.w3.org/TR/file-system-api
[5] http://www.w3.org/TR/file-writer-api

Ιστού θέλει να χρησιμοποιήσει μόνιμο σύστημα αρχείων, απαιτεί άδεια χρήστη και μόνο ένας χρήστης μπορεί να εγκρίνει ή να απορρίψει αυτό το αίτημα.

2.7. Υποστήριξη Προγράμματος Περιήγησης

Μόνο το Web Storage έχει κατάσταση Σύστασης W3C. Η ευρετηριασμένη βάση δεδομένων έχει την κατάσταση Υποψήφιας Σύστασης του W3C, ενώ η API αρχείου έχει την κατάσταση λειτουργίας προσχεδίου σε εξέλιξη(working draft). Για το λόγο αυτό, δεν υποστηρίζονται όλοι οι μηχανισμοί από διαφορετικά προγράμματα περιήγησης ιστού (Lonergan, 2012). Το Web Storage υποστηρίζεται από κάθε δημοφιλές σύγχρονο πρόγραμμα περιήγησης ιστού εκτός από το Opera Mini. Η IndexedDB δεν υποστηρίζεται από το Safari, το iOS Safari, το πρόγραμμα περιήγησης Android (υποστηρίζεται από την έκδοση 4.4 και μετά) και το Opera Mini (υποστηρίζεται από την έκδοση 10.0 ή νεότερη του Internet Explorer). Ορισμένες λειτουργίες του File API υποστηρίζονται από όλα τα προγράμματα περιήγησης εκτός από το Opera Mini. Ωστόσο, προς το παρόν υποστηρίζονται βασικά χαρακτηριστικά, όπως ένα σύστημα αρχείων sandbox και οι λειτουργίες CRUD σε αρχεία σε αυτό, αποκλειστικά από προγράμματα περιήγησης ιστού με μηχανή Webkit (Google Chrome, Opera, Blackberry Browser, Opera Mobile και Chrome για Android – αν και δεν υπάρχουν ενδείξεις εφαρμογής πλήρους υποστήριξης File API σε άλλα προγράμματα περιήγησης ιστού στο εγγύς μέλλον). Ο ακόλουθος **πίνακας** συνοψίζει τη διαθεσιμότητα των δυνατοτήτων HTML5 σε σύγχρονα προγράμματα περιήγησης ιστού.

	WebStorage	IndexedDB	File API	FileSystems, FileWriter
Internet Explorer	8.0	10.0	10,0	-
Mozilla Firefox	3.5	10.0	3.6	-
Google Chrome	4.0	23.0	13.0	13.0
Safari	4.0	-	6.0	-
Opera	10.5	15.0	11.1	15.0
iOS Safari	3.2	-	6.0	-
Opera Mini	-	-	-	-
Android Browser	2.1	-	3.0*	-
Blackberry Browser	7.0	10.0*	10.0	10.0
Opera Mobile	11.0	14.0	11.1	14.0
Google Chrome for Android	25.0	25.0	25.0	25.0
Firefox for Android	19.0	19.0	19.0	-

Πίνακας. Συμμόρφωση με τις δυνατότητες HTML5 για προγράμματα περιήγησης ιστού που διατίθενται στην αγορά

2.8. Εφαρμογές Web σε Φορητές Συσκευές

Λόγω των ειδικών περιορισμών των φορητών συσκευών και των πλαισίων ανάπτυξης που είναι διαθέσιμα για αυτές, οι περισσότερες εφαρμογές Ιστού που προορίζονται για φορητές συσκευές είναι εγγενείς εφαρμογές. Η πρόσβαση στο Διαδίκτυο από τέτοιες συσκευές είναι πολλές φορές αναξιόπιστη. Συνεπώς δεν μπορεί να θεωρηθεί δεδομένο ότι υπάρχει διαρκώς πρόσβαση στο διαδίκτυο. Επιπλέον, υπάρχουν και άλλες περιπτώσεις όπου οι συσκευές δεν μπορούν να συνδεθούν στο Διαδίκτυο, π.χ. κατά τη διάρκεια της πτήσης. Επομένως, οι εφαρμογές για κινητές συσκευές πρέπει να έχουν τη δυνατότητα να λειτουργούν τόσο σε σύνδεση (online) όσο και εκτός σύνδεσης (off-line). Ωστόσο, δουλεύοντας εκτός σύνδεσης δημιουργείται ζήτηση για αποθήκευση της κατάστασης της εφαρμογής στον χώρο αποθήκευσης των κινητών συσκευών. Δυστυχώς, μία τέτοια απαίτηση είναι πολύ δύσκολο να ικανοποιηθεί κατά τη χρήση προγραμμάτων περιήγησης ιστού σε κινητές συσκευές.

Υπάρχουν δύο κύριοι τύποι εφαρμογών για κινητά. Η πρώτη ονομάζεται "Mobile Web Applications". Αυτή αποτελείται από πρόγραμμα περιήγησης πελάτη – WWW και διακομιστή ιστού συνδεδεμένο σε μια πηγή δεδομένων. Η κατάσταση της εφαρμογής αποθηκεύεται στον διακομιστή. Για να χρησιμοποιήσει μια τέτοια εφαρμογή, ένας χρήστης πρέπει να δημιουργήσει σύνδεση στο Διαδίκτυο από την

συσκευή του και να ξεκινήσει ένα πρόγραμμα περιήγησης ιστού. Δεν απαιτείται εγκατάσταση κάποιου πρόσθετου λογισμικού. Οι προγραμματιστές εφαρμόζουν μόνο μία έκδοση μιας εφαρμογής Ιστού, η οποία είναι κατάλληλη για όλα τα σύγχρονα προγράμματα περιήγησης, είτε πρόκειται για προγράμματα περιήγησης ιστού για φορητές συσκευές, είτε για επιτραπέζιους υπολογιστές.

Ένας άλλος τύπος εφαρμογής για κινητά είναι ο «Έξυπνος πελάτης». Πρόκειται για μία εγγενή εφαρμογή, η οποία πρέπει να ληφθεί και να εγκατασταθεί από έναν χρήστη στη συσκευή του. Ως πλεονέκτημα, μια τέτοια εφαρμογή μπορεί να έχει πρόσβαση στον χώρο αποθήκευσης της συσκευής και να λειτουργεί εκτός σύνδεσης (π.χ. ορισμένες λειτουργίες εκτελούνται εκτός σύνδεσης και στη συνέχεια συγχρονίζονται όταν δημιουργηθεί η σύνδεση). Μια εγγενής εφαρμογή επιτρέπεται να έχει πρόσβαση στις λειτουργίες του λειτουργικού συστήματος μιας συσκευής, όπως γεωγραφική τοποθεσία, ανίχνευση διαθέσιμου καναλιού σύνδεσης (WiFi, 3G, κ.λπ.), επομένως μπορεί να προσφέρει πιο εξελιγμένες λειτουργίες. Ένα μεγάλο μειονέκτημα των εγγενών εφαρμογών είναι η ποικιλία των πλατφορμών για κινητές συσκευές που πρέπει να ληφθούν υπόψη από τους προγραμματιστές εφαρμογών (δηλαδή iOS, Android, Windows Phone, Blackberry). Ένας πάροχος εφαρμογών πρέπει να αναπτύξει πολλές εφαρμογές για να καλύψει όλους τους χρήστες κινητών συσκευών.

Για τους παραπάνω λόγους, έγιναν κάποιες προσπάθειες να επιταχυνθεί η διαδικασία ανάπτυξης εφαρμογών web σε αρχιτεκτονική thin-client για κινητές συσκευές, αυτές ονομάζονται υβριδικές εφαρμογές. Το PhoneGap είναι μία από αυτές (Cha & Yun, 2013). Αυτό το εργαλείο επιτρέπει σε έναν προγραμματιστή να δημιουργήσει μια εφαρμογή χρησιμοποιώντας HTML, Javascript και CSS. Από την άποψη ενός προγραμματιστή, η διαδικασία είναι παρόμοια με την ανάπτυξη μιας τυπικής εφαρμογής που βασίζεται στο web. Ωστόσο, μόλις ο πηγαίος κώδικας της εφαρμογής είναι έτοιμος, μεταγλωττίζεται σε μια εγγενή εφαρμογή αφιερωμένη σε κάθε πλατφόρμα για κινητά (χρησιμοποιώντας στοιχεία ελέγχου βάσει προβολών ιστού ειδικά για κάθε πλατφόρμα). Επομένως, η μεταγλωττισμένη εφαρμογή πρέπει να εγκατασταθεί σε μια συσκευή από έναν χρήστη και να υποβληθεί στη διαδικασία επικύρωσης πριν τοποθετηθεί στην αγορά της πλατφόρμας (δηλαδή Play Store για Android ή App Store σε iOS).

Μια άλλη προσέγγιση είναι η RWD (Responsive Web Design). Επειδή μερικές φορές είναι δύσκολο να προετοιμαστεί μια ιστοσελίδα που να φαίνεται σωστά τόσο σε μεγάλη οθόνη Full-HD όσο και σε σχετικά μικρή οθόνη κινητής συσκευής, οι προγραμματιστές τείνουν να επιλέγουν τη δημιουργία εγγενών εφαρμογών προσαρμοσμένων στο μέγεθος οθόνης (μερικές φορές πρόκειται για διαφορετικό λογισμικό για τηλέφωνα και tablet). Το HTML5 εισάγει ερωτήματα διάστασης της οθόνης (media queries) – στοιχεία του CSS που μπορούν να χρησιμοποιηθούν από προγραμματιστές για να προσαρμόσουν την εμφάνιση της ιστοσελίδας ανάλογα με το μέγεθος του παραθύρου του προγράμματος περιήγησης Ιστού.

Φαίνεται ότι, λόγω της διαθεσιμότητας της HTML5 και των παραπάνω τεχνολογιών, οι προγραμματιστές έχουν μεγάλες ευκαιρίες να ενισχύσουν τις εφαρμογές web τους σε αρχιτεκτονική thin client. Η αποθήκευση δεδομένων στην πλευρά του πελάτη μπορεί εύκολα να επιτευχθεί χρησιμοποιώντας το Web Storage, την IndexedDB και την File API ακόμη και με σχετικά μεγάλα σύνολα δεδομένων και δυαδικά αρχεία. Σίγουρα, υπάρχουν ορισμένοι τύποι εφαρμογών για κινητά που πρέπει ακόμα να είναι εγγενείς, π.χ. εφαρμογές που τρέχουν πολύ γρήγορα, αλλά οι περισσότερες μπορούν να παρέχονται ως HTML5, εφαρμογές για πελάτες που βασίζονται στον ιστό με χώρο αποθήκευσης εκτός σύνδεσης από την πλευρά του πελάτη. Αυτές οι εφαρμογές μπορούν να εκτελεστούν και στις δύο λειτουργίες: online και offline.

2.9. Πιθανή Υιοθέτηση Νέων Τεχνολογιών W3C

Όπως ισχυρίζεται ο οργανισμός W3C, υπάρχουν πολλές χρήσιμες περιπτώσεις που σχετίζονται με τις αναφερόμενες τεχνολογίες που πρέπει να υιοθετηθούν σε εφαρμογές web. Μία τέτοια εφαρμογή μπορεί να είναι μια μονάδα που στέλνει μεγάλα σύνολα δεδομένων από ένα πρόγραμμα περιήγησης ιστού πελάτη σε έναν διακομιστή ιστού. Το σύνολο δεδομένων χωρίζεται σε μικρότερα κομμάτια, τα οποία αποθηκεύονται στο χώρο ενός προγράμματος περιήγησης και αποστέλλονται διαδοχικά. Εάν ένα παράθυρο του προγράμματος περιήγησης ιστού είναι κλειστό, τότε η εφαρμογή θυμάται το τελευταίο τμήμα που απεστάλη. Μετά την επόμενη πρόσβαση στην ιστοσελίδα, αποστέλλονται και τα υπόλοιπα μέρη. Χωρίς δυνατότητα αποθήκευσης δεδομένων σε πρόγραμμα περιήγησης Ιστού, σε περίπτωση διακοπής μεταφοράς (ανεξαρτήτως αιτίας) ένα σύνολο δεδομένων θα πρέπει να μεταφορτωθεί ξανά και να σταλεί από την αρχή.

Ένα άλλο σενάριο αφορά επίσης την αποστολή αρχείων. Έστω ότι μια εφαρμογή που εκτελείται σε ένα πρόγραμμα περιήγησης ιστού χρειάζεται να κατεβάσει πολλά αρχεία από έναν διακομιστή. Είναι πιο γρήγορο να μεταφερθεί ένα πακέτο ZIP από πολλά αρχεία ξεχωριστά. Σε αυτήν την περίπτωση, μετά τη λήψη του αιτήματος HTTP του πελάτη, πολλά αρχεία μπορούν να συμπιεστούν στον διακομιστή και να σταλούν πίσω ως ένα ενιαίο πακέτο. Στη συνέχεια, το πακέτο μπορεί να αποσυμπιεστεί στο πρόγραμμα περιήγησης ιστού και το περιεχόμενό του μπορεί να αποθηκευτεί σε χώρο αποθήκευσης του προγράμματος περιήγησης ιστού για πρόσβαση από αυτό (επίσης εκτός σύνδεσης).

Η επόμενη περίπτωση είναι ο επεξεργαστής γραφικών εκτός σύνδεσης που εκτελείται μόνο σε πρόγραμμα περιήγησης ιστού χωρίς να χρειάζεται να αποσταλούν αρχεία σε διακομιστή. Επιπλέον, τα αρχεία μπορούν να αποθηκευτούν στον χώρο αποθήκευσης ενός προγράμματος περιήγησης Ιστού για περαιτέρω χειρισμό. Στην περίπτωση κινητών συσκευών, αυτό είναι ένα παράδειγμα χρήσιμης εφαρμογής που θα μπορούσε να αντικαταστήσει με επιτυχία το εγγενές λογισμικό.

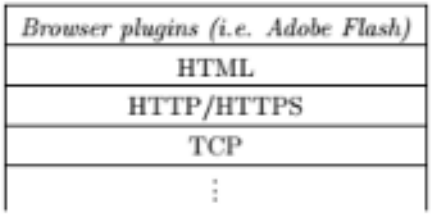
Ένα άλλο παράδειγμα είναι μια δικτυακή πύλη που παρέχει πρόσβαση σε ταινίες. Τώρα, η πύλη μπορεί να βελτιστοποιηθεί με τη δυνατότητα αποθήκευσης ταινιών σε προγράμματα περιήγησης, ώστε να μπορούν να παρακολουθούνται όταν δεν υπάρχει διαθέσιμη σύνδεση στο Διαδίκτυο λόγω προβλημάτων δικτύου ή πτήσης.

Όταν χρησιμοποιούνται νέες τεχνολογίες HTML5, είναι δυνατή η εφαρμογή ενός προγράμματος-πελάτη ηλεκτρονικού ταχυδρομείου που βασίζεται στον ιστό που επιτρέπει την πρόσβαση σε περιεχόμενο ηλεκτρονικού ταχυδρομείου εκτός σύνδεσης (συμπεριλαμβανομένων των συνημμένων που έχουν προηγουμένως ληφθεί και αποθηκευτεί στον χώρο αποθήκευσης ενός προγράμματος περιήγησης Ιστού). Ο χρήστης έχει τη δυνατότητα να χρησιμοποιεί την εφαρμογή σε λειτουργία εκτός σύνδεσης, αλλά ταυτόχρονα είναι σε θέση να μειώνει το φόρτο του δικτύου όταν εργάζεται στο διαδίκτυο. Μια άλλη σημαντική δυνατότητα που είναι χρήσιμη εκτός σύνδεσης είναι η αποθήκευση αντιγράφων που λειτουργούν, καθώς και συνημμένων, ως πρόχειρα και τα οποία θα συγχρονιστούν με τον διακομιστή μόλις ο χρήστης συνδεθεί ξανά στο διαδίκτυο (όταν επανασυνδεθεί το Διαδίκτυο, τα στοιχεία θα ληφθούν από το χώρο του προγράμματος περιήγησης ιστού και θα σταλούν ως μήνυμα ηλεκτρονικού ταχυδρομείου).

3. Τεχνικές Αποθήκευσης σε HTML5

Το 2014 το W3C δημοσίευσε την επίσημη σύσταση HTML5 (Hickson et al., 2014). Αυτή η νέα προδιαγραφή έγινε για να βελτιωθεί η υποστήριξη για διαδραστικές εφαρμογές Ιστού. Το νέο πρότυπο εισήγαγε επίσης ορισμένες νέες τεχνικές που μπορούν να χρησιμοποιηθούν για την παρακολούθηση των χρηστών. Το κεφάλαιο αυτό επικεντρώνεται στις λειτουργίες αποθήκευσης εντός της προδιαγραφής HTML5, δηλαδή την τοπική αποθήκευση (Local Storage) και την IndexedDB και αναλύει τον τρόπο λειτουργίας τους και τις δυνατότητες παρακολούθησης. Επίσης γίνεται ανάλυση για τον τρόπο με τον οποίο χρησιμοποιούνται αυτές οι τεχνικές ενώ δεν εξετάστηκαν τα δακτυλικά αποτυπώματα (fingerprinting) και οι τεχνικές παρακολούθησης βάσει κρυφής μνήμης που περιλαμβάνονται στην HTML 5.

Η HTML (Hypertext Markup Language) είναι η γλώσσα που χρησιμοποιείται για τον καθορισμό του περιεχομένου και της διάταξης των ιστότοπων. Τα προγράμματα περιήγησης Ιστού μπορούν στη συνέχεια να αποδώσουν τον ιστότοπο και να τον εμφανίσουν στον χρήστη με τον τρόπο που πρέπει να μοιάζει. Η HTML χρησιμοποιείται συνήθως πάνω από το HTTP. Οι δυνατότητες που δεν υποστηρίζονται από την επιλεγμένη έκδοση HTML μπορούν να προστεθούν χρησιμοποιώντας πρόσθετα. Το Adobe Flash και το Microsoft Silverlight είναι παραδείγματα τέτοιων προσθέτων. Η αποτύπωση του παραπάνω φαίνεται στο ακόλουθο διάγραμμα:



Διάγραμμα: Η πιο κοινή στοίβα πρωτοκόλλων Ιστού

Τα πρόσθετα προγράμματος περιήγησης τοποθετούνται στην κορυφή της στοίβας. Η αυστηρά εμφανής επικοινωνία των προσθηκών προγράμματος περιήγησης διαχωρίζεται πλήρως από τη στοίβα πρωτοκόλλου Ιστού, επειδή η επικοινωνία πελάτη-διακομιστή των προσθέτων δεν περιλαμβάνεται σε HTML. Πρέπει να γίνει αντιληπτό ότι γενικά τα πρόσθετα του προγράμματος περιήγησης βασίζονται σε

HTML και χρειάζονται μόνο όταν λείπει η συγκεκριμένη λειτουργικότητα στην υποκείμενη δομή.

Με τα cookies HTTP οι χρήστες μπορούσαν να αναγνωριστούν μόνο κατόπιν επαφής τους με τον αντίστοιχο διακομιστή. Στην HTML5 αυτό δεν αποτελεί πλέον περιορισμό, επειδή αυτά τα στοιχεία είναι διαθέσιμα μέσω JavaScript, επομένως τα σενάρια εκτός σύνδεσης μπορούν επίσης να παρακολουθούν τη συμπεριφορά των χρηστών μεταξύ αυτών των στιγμών. Αυτές οι πληροφορίες μπορούν στη συνέχεια να αποσταλούν στο σημείο παρακολούθησης. Κάτι τέτοιο ήταν δυνατό μέχρι τώρα μόνο με σενάρια που εκτελούνται σε μια σελίδα και τα οποία έστελναν αμέσως τις πληροφορίες που συνέλεγαν σε ένα μέρος παρακολούθησης το οποίο όμως συνεπάγεται μεγαλύτερα έξοδα για το μέλος παρακολούθησης ενώ η περισσότερη παρακολούθηση χρηστών σε πραγματικό χρόνο δίνει στα μέρη παρακολούθησης την ευκαιρία να προσαρμόσουν τον ιστότοπό τους εν κινήσει (δηλαδή να αλλάξουν διαφημίσεις).

Λαμβάνοντας υπόψη τη στοίβα πρωτοκόλλου που απεικονίστηκε παραπάνω, τα cookie HTTP είναι πάντα παρόντα ως εναλλακτική λύση για τις τεχνικές αποθήκευσης HTML5. Αυτό συμβαίνει επειδή το HTTP είναι απαραίτητο ως υποκείμενο πρωτόκολλο για HTML ενώ δεν υπάρχει καμία πρόθεση ώστε αυτό να αλλάξει λόγω της συμβατότητας προς τα πίσω στη νέα έκδοση του HTTP (HTTP/2).

3.1. Τοπική Αποθήκευση (LocalStorage)

Η πιο γνωστή τεχνική αποθήκευσης HTML5 είναι η τοπική αποθήκευση (Local Storage) που περιλαμβάνεται στη Web Storage API. Αυτή η API παρέχει παρόμοια τεχνική με τα cookie HTTP. Τα αντικείμενα αποθήκευσης υλοποιούνται ως ζεύγη κλειδιών/τιμών. Οι ιστότοποι μπορούν να χρησιμοποιήσουν πολλά αντικείμενα για να αποθηκεύσουν τις πληροφορίες που χρειάζονται και αργότερα οι πληροφορίες μπορούν να ανακτηθούν ζητώντας το αντίστοιχο κλειδί αναγνώρισης. Η λειτουργικότητα που παρέχει η διεπαφή αποθήκευσης είναι παρόμοια με τα cookie HTTP. Επειδή το HTML βασίζεται σε HTTP, οι προγραμματιστές συνήθως έχουν και τις δύο τεχνικές διαθέσιμες συνεχώς.

Υπάρχουν δύο υλοποιήσεις της διεπαφής αποθήκευσης που μπορούν να χρησιμοποιηθούν:

Η τοπική αποθήκευση είναι η υλοποίηση που χρησιμοποιείται για μόνιμη αποθήκευση δεδομένων. Υπάρχουν ορισμένες σημαντικές διαφορές μεταξύ των cookie HTTP και της Local Storage. Αρχικά, τα αντικείμενα Local Storage δεν γνωρίζουν ημερομηνία λήξης. Επομένως, αποθηκεύονται για πάντα, εκτός εάν διαγραφούν ρητά κάποια στιγμή. Δεύτερον, τα cookie HTTP αποστέλλονται αυτόματα σε έναν διακομιστή ιστού σε όλα τα αιτήματα που κάνει ένα πρόγραμμα περιήγησης ιστού. Στην τοπική αποθήκευση, οι πόροι πρέπει να ζητηθούν με μη αυτόματο τρόπο μέσω JavaScript. Για να μπορεί ένας διακομιστής web να λάβει αυτούς τους πόρους, πρέπει αυτοί να συμπεριληφθούν σε ένα αίτημα προς τον διακομιστή χειροκίνητα. Αυτό απαιτεί περισσότερη επιβάρυνση στον κώδικα JavaScript.

Η αποθήκευση πληροφοριών στο Local Storage είναι αρκετά εύκολη. Με ορισμένες απλές δηλώσεις JavaScript, οι πληροφορίες μπορούν να αποθηκευτούν μόνιμα στο πρόγραμμα περιήγησης.

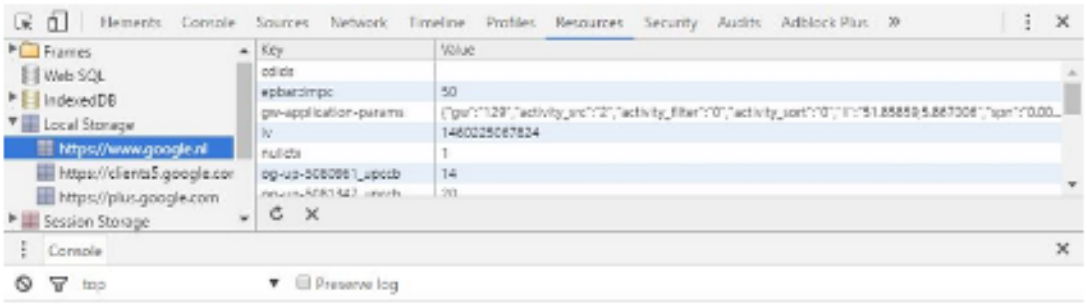
/ / Storing information with key ” test ”

local Storage . test = ” Hello world ! ” ;

/ / Retrieving information

varx = local Storage. test ;

Οι πληροφορίες που είναι αποθηκευμένες στον Τοπικό χώρο αποθήκευσης μπορούν να προβληθούν στο development kit των περισσότερων σύγχρονων προγραμμάτων περιήγησης (π.χ. Google Chrome, Mozilla Firefox και Microsoft Edge). Ένα παράδειγμα αυτής της προβολής στο Google Chrome φαίνεται στην **εικόνα** που ακολουθεί



Εικόνα: Τοπικοί πόροι αποθήκευσης του google.nl που εμφανίζονται στο Google Chrome

Χρησιμοποιείται για την αποθήκευση δεδομένων στην πλευρά του πελάτη. Δεν έχει χρόνο λήξης, επομένως τα δεδομένα στο Local Storage υπάρχουν πάντα έως ότου ο χρήστης τα διαγράψει με μη αυτόματο τρόπο.

Σύνταξη:

- Για αποθήκευση δεδομένων σε χώρο αποθήκευσης Ιστού: Το κλειδί και η τιμή πρέπει να είναι συμβολοσειρά ή αριθμός.

```
LocalStorage.setItem("key", "value");
```

- Για λήψη δεδομένων από την αποθήκευση ιστού: Δίνεται το κλειδί και επιστρέφεται η τιμή.

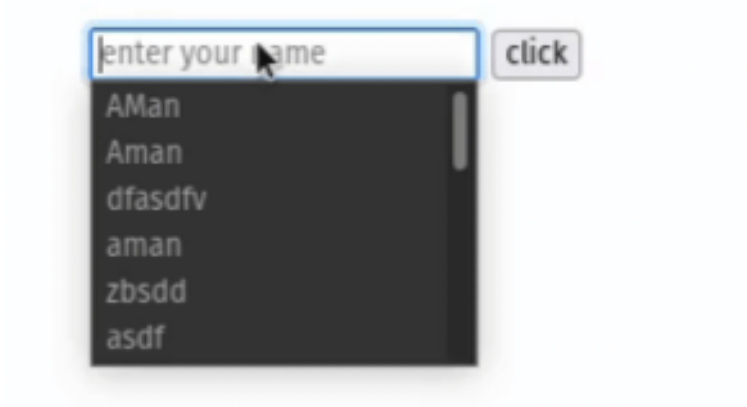
```
LocalStorage.getItem("key");
```

Παράδειγμα:

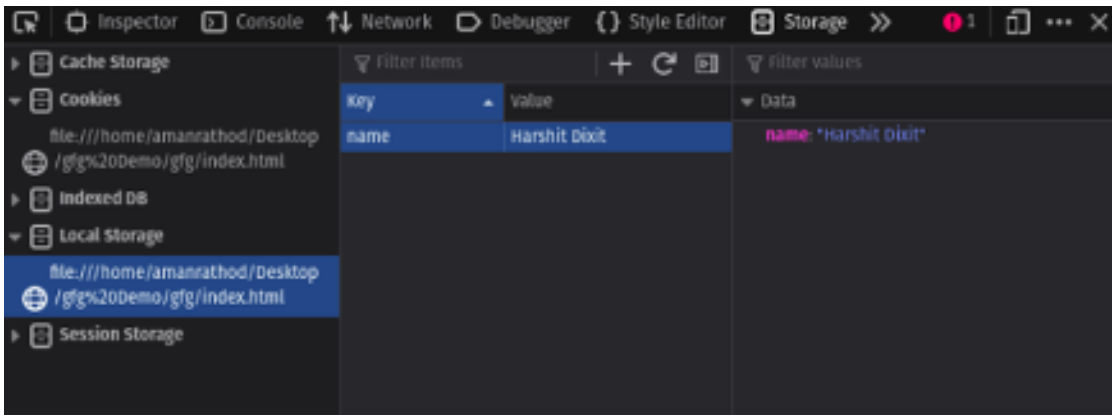
```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        color: green;
        text-align: center;
        font-size: 30px;
        margin-top: 30px;
        font-style: italic;
      }
      #data {
        text-align: center;
      }
    </style>
  </head>
  <body>
    <input id="name" type="name" placeholder="enter your name" />
    <button type="submit" onClick="handleClick()">click</button>
    <br />
    <div id="data"></div>

    <script>
      function handleClick() {
        if (typeof Storage !== "undefined") {
          let name = document.getElementById("name").value;
          localStorage.setItem("name", name);
```

```
          document.getElementById("data").innerHTML =
            "Welcome To GeeksforGeeks" + " " + localStorage.name;
        } else {
          alert("Sorry! your browser doesn't support Web
Storage");
        }
      }
    </script>
  </body>
</html>
```

Έξοδος:



Αποθηκευμένα δεδομένα στον Τοπικό χώρο αποθήκευσης



Τα στοιχεία τοπικής αποθήκευσης αποθηκεύονται με τη μορφή ζεύγους κλειδιών/τιμών και μπορούν να υποστούν έλεγχο απλώς επιθεωρώντας στοιχεία στην ιστοσελίδα. Εν συνεχεία πρέπει να γίνει μετάβαση στην επιλογή Application όπου υπάρχει ο τοπικός χώρος αποθήκευσης.

Το αντικείμενο local Storage αποθηκεύει τα δεδομένα χωρίς ημερομηνία λήξης, κάτι που μπορεί εύκολα να διαπιστωθεί με κλείσιμο της τρέχουσας καρτέλας και

επίσκεψη ξανά της ίδιας σελίδας, οπότε και διαπιστώνει κανείς ότι τα ίδια δεδομένα υπάρχουν στο local Storage αυτής της καρτέλας ή παραθύρου.

3.2. Αποθήκευση περιόδου λειτουργίας (SessionStorage)

Αυτή η τεχνική αποθηκεύει πληροφορίες μόνο για μία περίοδο λειτουργίας προγράμματος περιήγησης. Η ιδέα αυτού είναι παρόμοια με την ιδέα των μη μόνιμων cookie HTTP. Η μόνη διαφορά είναι ότι η αποθήκευση περιόδου λειτουργίας περιορίζεται πραγματικά στο πλαίσιο μιας συγκεκριμένης περιόδου λειτουργίας. Τα μη μόνιμα cookie HTTP, αντίθετα, μοιράζονται μεταξύ πολλών καρτελών του ίδιου ιστότοπου και επομένως δεν περιορίζονται σε μία περίοδο λειτουργίας.

Για σκοπούς παρακολούθησης, η αποθήκευση συνεδρίας μπορεί να χρησιμοποιηθεί μόνο για την παρακολούθηση ενός χρήστη σε έναν ιστότοπο κατά τη διάρκεια μιας συγκεκριμένης περιόδου λειτουργίας. Επομένως, αυτή η συγκεκριμένη τεχνική δεν είναι τόσο χρήσιμη ως αυτόνομη τεχνική παρακολούθησης. Για το λόγο αυτό τώρα πια καθίσταται σημαντική μόνο η τοπική αποθήκευση.

Χρησιμοποιείται για την αποθήκευση δεδομένων στην πλευρά του πελάτη. Τα δεδομένα στο SessionStorage υπάρχουν για όσο μένει ανοιχτή η τρέχουσα καρτέλα. Κλείσιμο της τρέχουσας καρτέλας συνεπάγεται και αυτόματη διαγραφή των δεδομένων από το SessionStorage.

Σύνταξη:

- Για αποθήκευση δεδομένων σε χώρο αποθήκευσης ιστού:

```
SessionStorage.setItem("key", "value");
```

- Για λήψη δεδομένων από την αποθήκευση ιστού:

```
SessionStorage.getItem("key");
```

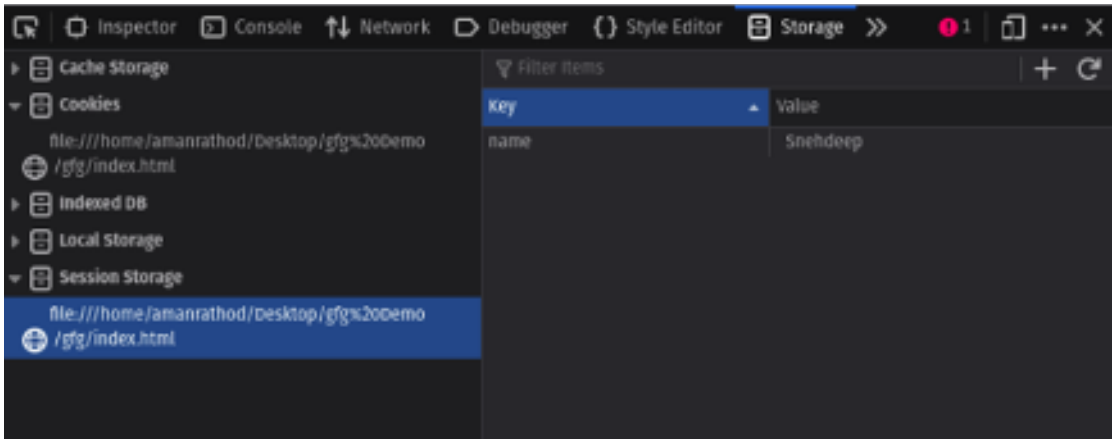
Παράδειγμα :

```
<!DOCTYPE html>
<html>
<head>
  <style>
      body {
        color: green;
        text-align: center;
        font-size: 30px;
        margin-top: 30px;
```

```
        font-style: italic;
      }
      #data {
        text-align: center;
      }
  </style>
</head>
<body>
    <input id="name" type="name" placeholder="enter your name" >
    <button type="submit" onClick="handleClick()">click</button>
    <br>
    <div id="data"></div>

    <script>
      function handleClick() {
        if(typeof(Storage)!=="undefined") {
          let name = document.getElementById("name").value;
            sessionStorage.setItem("name", name);

            document.getElementById("data").innerHTML =
              ("Welcome To GeeksforGeeks"+"
"+sessionStorage.name);
          }
          else{
            alert("Sorry! your browser is not supporting the
browser")
          }
       }
    </script>
</body>
</html>
```

Έξοδος:

Αποθηκευμένα δεδομένα στο Session Storage :



Το αντικείμενο session Storage αποθηκεύει τα δεδομένα με την ημερομηνία λήξης τους. Ο χρήστης για να το διαπιστώσει μπορεί κλείσει την τρέχουσα καρτέλα και να επισκεφθεί ξανά την ίδια σελίδα, οπότε και θα διαπιστώσει ότι τα δεδομένα είναι κενά στην session Storage λειτουργίας της συγκεκριμένης καρτέλας ή παραθύρου.

3.3. Αποθήκευση με Χρήση Βάσεων Δεδομένων

Η Κοινοπραξία του Παγκόσμιου Ιστού (World Wide Web Consortium-W3C) τυποποίησε την HTML5 για την επόμενη γενιά εφαρμογών Ιστού. Με το νέο πρότυπο HTML5 αναμένονται νέες λειτουργίες μεταξύ των οποίων περιλαμβάνεται η IndexedDB , η οποία είναι μια βάση δεδομένων που βασίζεται σε πρόγραμμα περιήγησης από την πλευρά του πελάτη. Η ανάγκη για IndexedDB αντικατοπτρίζει τις απαιτήσεις για περισσότερο χώρο αποθήκευσης, ο οποίος παραμένει πέρα από τις ανανεώσεις της σελίδας, ενώ αποφεύγεται η μεταφορά δεδομένων στον διακομιστή. Επίλυση στο ζήτημα μπορεί να αποτελέσει η αποθήκευση των δεδομένων στον υπολογιστή-πελάτη. Οι παραδοσιακές σχεσιακές βάσεις δεδομένων SQL χρησιμοποιούνται από το 1976 (Chamberlin, 1976). Οι αλλαγές που έκαναν ορισμένες εταιρείες ήταν να χρησιμοποιήσουν αντικειμενοστρεφείς βάσεις δεδομένων, καθώς αυτές έχουν ορισμένα πλεονεκτήματα σε σχέση με την SQL. Το κίνητρο για τις αλλαγές και τις βελτιώσεις που έρχονται με την HTML5 είναι ότι το πρόγραμμα περιήγησης ιστού θα πρέπει να μπορεί να εκτελεί εφαρμογές από την πλευρά του πελάτη με τον ίδιο τρόπο που μπορεί να εκτελεί εφαρμογές επιτραπέζιου υπολογιστή. Δηλαδή, η διαδικασία από την πλευρά του πελάτη θα πρέπει να είναι σε θέση να αποφύγει την αναποτελεσματικότητα και τα προβλήματα συνδεσιμότητας δικτύου που εντοπίζονται σε εφαρμογές από την πλευρά του διακομιστή. Κατά συνέπεια, τα μεγάλα προγράμματα περιήγησης υποστηρίζουν πλέον τα περισσότερα από τα νέα στοιχεία HTML5 και τις διεπαφές προγραμματισμού εφαρμογών (API).

Επομένως, μια βάση δεδομένων από την πλευρά του προγράμματος περιήγησης HTML5 μπορεί να περιέχει αποθηκευμένα δεδομένα από διαδικτυακές υπηρεσίες που κάνουν χρήση της νέας λειτουργικότητας της HTML5. Παραδοσιακές εφαρμογές επιτραπέζιου υπολογιστή, όπως επεξεργαστές κειμένου και υπολογιστικά φύλλα, μπορούν να χρησιμοποιηθούν με εφαρμογές Ιστού. Αυτό σημαίνει ότι ο πελάτης δεν θα χρειαστεί να εγκαταστήσει οποιοδήποτε λογισμικό στον υπολογιστή, αλλά θα χρειαστεί μόνο ένα πρόγραμμα περιήγησης στο Web. Καθώς η πρόσβαση στο Διαδίκτυο είναι ζωτικής σημασίας για τις εφαρμογές Ιστού, η χρήση αυτών των εφαρμογών Ιστού θα πρέπει να είναι δυνατή ανεξάρτητα από τη σύνδεση, έτσι οι εφαρμογές εκτός σύνδεσης αποτελούν σημαντικό μέρος αυτής της ανάπτυξης.

Η εργασία θα συγκρίνει διάφορες βάσεις δεδομένων σε διαφορετικά περιβάλλοντα προγράμματος περιήγησης. Τόσο οι βάσεις δεδομένων του πελάτη όσο και του διακομιστή ελέγχονται. Οι βάσεις δεδομένων από την πλευρά του πελάτη περιλαμβάνουν το IndexedDB, το LocalStorage και το καταργημένο WebSQL. Οι βάσεις δεδομένων από την πλευρά του διακομιστή περιλαμβάνουν MySQL με InnoDB και MyISAM. Η συγκριτική αξιολόγηση περιέχει δεδομένα από διάφορα σύνολα δοκιμών, τα οποία περιλαμβάνουν τις κοινές λειτουργίες δημιουργίας, ανάγνωσης, ενημέρωσης και διαγραφής δεδομένων (CRUD) από/προς μια βάση δεδομένων.

Η IndexedDB υλοποιείται με διαφορετικό τρόπο στα διαφορετικά προγράμματα περιήγησης. Ο Firefox χρησιμοποιεί το SQLite και το Chrome χρησιμοποιεί τη LevelDB (η οποία δεν είναι βάση δεδομένων SQL). Όπως και οι NoSQL και Dbm, δεν διαθέτει μοντέλο σχεσιακών δεδομένων, δεν υποστηρίζει ερωτήματα SQL και δεν υποστηρίζει ευρετήρια, επομένως, ακόμη κι αν η IndexedDB έχει δημιουργηθεί στον Firefox, χρησιμοποιείται στην πραγματικότητα μια τεχνολογία που υποστηρίζεται από SQL με επιβάρυνση που μοιάζει με SQL.

3.3.1. Σχεσιακή βάση δεδομένων

Τα σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων (relational Database Management Systems-RDMS) αντιπροσωπεύουν εγγραφές οργανωμένες σε πίνακες. Η δομή των πινάκων αποτελείται από στήλες και γραμμές. Οι στήλες αντιπροσωπεύουν κατηγορίες δεδομένων και οι σειρές τα δεδομένα. Η δομή των σχεσιακών βάσεων δεδομένων είναι καλή για τη διαχείριση μεγάλου όγκου

δομημένων δεδομένων. Το μειονέκτημα είναι η ακαμψία τους, γιατί η μοναδική δομή δεδομένων τους είναι οι πίνακες. Παρουσιάζουν πρόβλημα όσον αφορά το χειρισμό πολύπλοκων αρχείων πολυμέσων, κάτι που σήμερα είναι σημαντικό για πολύπλοκες διαδικτυακές εφαρμογές (Harrington, 2).

Οι σχεσιακές βάσεις δεδομένων είναι προγράμματα υπολογιστών που χρησιμοποιούνται για την αποθήκευση πληροφοριών σε πίνακες. Αυτοί οι πίνακες περιέχουν γραμμές και στήλες που χρησιμοποιούνται για την ταξινόμηση και την ανάκτηση πληροφοριών. Οι σειρές και οι στήλες περιέχουν πληροφορίες σχετικά με το θέμα του πίνακα. Ο διαχειριστής της βάσης δεδομένων μπορεί να ορίσει τις σχέσεις μεταξύ των διαφόρων τύπων δεδομένων. Οι σχεσιακές βάσεις δεδομένων απαιτούν τα δεδομένα να εισάγονται με τη μορφή ακεραίων, συμβολοσειρών ή πραγματικών αριθμών. Αυτά τα δεδομένα πρέπει στη συνέχεια να προσπελαστούν μέσω ερωτημάτων SQL (Conolly & Begg, 2004).

Ένα διάγραμμα σχέσεων οντοτήτων (entity relationship diagram-ERD) αποτελεί μια χρήσιμη τεχνική για τη διαχείριση της ανάπτυξης ενός συστήματος πληροφοριών βάσης δεδομένων. Το ERD μοντελοποιεί δεδομένα σε λογικές και εύκολα κατανοητές γραφικές αναπαραστάσεις (Thalheim, 2013). Τα διαγράμματα σχέσεων οντοτήτων απεικονίζουν τη λογική δομή των βάσεων δεδομένων. Τα κουτιά χρησιμοποιούνται για την αναπαράσταση οντοτήτων, τα διαμάντια χρησιμοποιούνται συνήθως για την αναπαράσταση σχέσεων και τα οβάλ χρησιμοποιούνται για την αναπαράσταση ιδιοτήτων. Η σχέση μεταξύ των πλαισίων μπορεί να είναι:

• Ενα προς ένα

• Ένα προς πολλά

• Πολλά προς πολλά

Οι σχέσεις από το ERD πρόκειται να χρησιμοποιηθούν κατά τη δημιουργία των πινάκων της βάσης δεδομένων για τη δημιουργία σχέσεων μεταξύ τους. Έτσι το διάγραμμα δείχνει τη σχέση μεταξύ των αντικειμένων.

3.3.2. Βάση δεδομένων SQLlite από την πλευρά του διακομιστή

Η SQLite είναι η πιο ευρέως χρησιμοποιούμενη βάση δεδομένων για ιστοσελίδες σύμφωνα με έρευνα της Netcraft. Το κύριο πλεονέκτημα της είναι η διαθεσιμότητα (χρησιμοποιείται από προγράμματα περιήγησης για κινητά σε Android και iOS, από PhoneGap για εφαρμογές για κινητά και από Chrome 15 και Safari 5).

Το κύριο μειονέκτημα της SQLite είναι ότι ορισμένα προγράμματα περιήγησης, όπως ο Firefox, αφαίρεσαν την υποστήριξη SQLite στις πιο πρόσφατες εκδόσεις τους. Η ενσωμάτωση της SQLite σε προγράμματα περιήγησης ιστού είχε ως αποτέλεσμα την προσθήκη της στο πρότυπο αποθήκευσης Ιστού HTML5 και μετά από συζήτηση εντός της Ομάδας Εργασίας Εφαρμογών Ιστού του W3C.

3.3.3. Βάση δεδομένων NoSQL από την πλευρά του πελάτη

Το NoSQL (Όχι μόνο SQL) (Strozzi, 1998) αποτελεί τη λύση για βάση δεδομένων που δεν είναι σχεσιακή ή αντικειμενοστραφής. Αποθηκεύει δεδομένα σε μορφή κλειδιού/τιμής. Η βάση δεδομένων μπορεί να χειριστεί μεγάλο όγκο δεδομένων, όπου δεν χρειάζεται σχεσιακό μοντέλο. Στην πράξη χρησιμοποιήθηκαν μόνο τη στιγμή που οι σχεδιαστές διαδικτυακών υπηρεσιών με πολύ μεγάλο αριθμό χρηστών ανακάλυψαν ότι τα παραδοσιακά σχεσιακά συστήματα διαχείρισης βάσεων δεδομένων (RDBMS) είναι κατάλληλα είτε για μικρές αλλά συχνές συναλλαγές ανάγνωσης/εγγραφής είτε για μεγάλες μαζικές συναλλαγές με σπάνια εγγραφή προσβάσεις και φυσικά όχι για μεγάλους φόρτους εργασίας ανάγνωσης/εγγραφής (κάτι που συμβαίνει συχνά σε αυτές τις υπηρεσίες web μεγάλης κλίμακας όπως το Google, το Amazon, το Facebook, το Yahoo και άλλες) (Tudorica & Bucur, 2011).

Οι βάσεις δεδομένων NoSQL γενικά επεξεργάζονται δεδομένα πιο γρήγορα από τις σχεσιακές βάσεις δεδομένων (Leavitt, 2010). Οι σχεσιακές βάσεις δεδομένων χρησιμοποιούνται συνήθως από επιχειρήσεις και συχνά για συναλλαγές που απαιτούν μεγάλη ακρίβεια. Οι προγραμματιστές συνήθως δεν χρησιμοποιούν τις βάσεις δεδομένων NoSQL που υποστηρίζουν ACID ((atomicity, consistency, isolation, durability/(ατομικότητα, συνέπεια, απομόνωση, ανθεκτικότητα), προκειμένου να αυξήσουν την απόδοση. Ωστόσο αυτό μπορεί να προκαλέσει προβλήματα όταν χρησιμοποιούνται για εφαρμογές που απαιτούν μεγάλη ακρίβεια. Οι βάσεις δεδομένων NoSQL είναι επίσης συχνά πιο γρήγορες επειδή τα μοντέλα δεδομένων τους είναι πιο απλά. Σύμφωνα με τον (Leavitt,2010) υπάρχει μια αντιστάθμιση

μεταξύ της ταχύτητας και της πολυπλοκότητας του μοντέλου, αλλά συχνά πρόκειται για ένα συμβιβασμός που αξίζει να γίνει.

Οι βάσεις δεδομένων NoSQL αντιμετωπίζουν πολλές προκλήσεις, οι πιο σημαντικές εκ των οποίων είναι η επιβάρυνση και η πολυπλοκότητα. Δεν λειτουργούν με ερωτήματα SQL, πράγμα που σημαίνει ότι πρέπει να προγραμματιστούν χειροκίνητα. Σε περιπτώσεις απλών εργασιών εκτελούνται γρήγορα, αλλά είναι χρονοβόρες στην περίπτωση των πολύπλοκων ερωτημάτων (Leavitt, 2010).

Οι σχεσιακές βάσεις δεδομένων υποστηρίζουν εγγενώς τις ACID (Conolly & Begg, 2004), ενώ οι βάσεις δεδομένων NoSQL όχι τόσο, το αποτέλεσμα είναι οι τελευταίες να μην προσφέρουν αξιοπιστία. Για την εκτέλεση αυτής της λειτουργίας απαιτείται πρόσθετος προγραμματισμός.

Η έλλειψη υποστήριξης των συναλλαγών ACID οδηγεί σε συμβιβασμούς στο κομμάτι της συνέπειας. Οι τραπεζικοί ιστότοποι χρησιμοποιούν τη συνέπεια στις εφαρμογές τους. Επομένως, η χρήση βάσεων δεδομένων NoSQL μπορεί να αποτελέσει πρόβλημα (Shashank, 2011). Από την άλλη παρέχουν καλύτερη απόδοση και επεκτασιμότητα. Οι περισσότεροι οργανισμοί δεν είναι εξοικειωμένοι με τις βάσεις δεδομένων NoSQL και έτσι μπορεί να μην αισθάνονται ικανοί γνώστες ώστε να επιλέξουν μία ή ακόμη και να καθορίσουν ότι η προσέγγιση μπορεί να είναι καλύτερη για τους σκοπούς τους (Stonebraker, 2010). Σε αντίθεση με τις εμπορικές σχεσιακές βάσεις δεδομένων, πολλές εφαρμογές NoSQL ανοιχτού κώδικα δεν διαθέτουν ακόμη εργαλεία υποστήριξης πελατών ή διαχείρισης. Κάθε βάση δεδομένων NoSQL έχει το δικό της σύνολο API, βιβλιοθήκες και προτιμώμενες γλώσσες για την αλληλεπίδραση με τα δεδομένα που περιέχουν.

Λίγα παραδείγματα βάσης δεδομένων NoSQL που προσανατολίζονται σε έγγραφα περιλαμβάνουν τις MongoDB, LevelDB, BerkleyDB. Οι δύο πρώτες αποθηκεύουν τα δεδομένα στον σκληρό δίσκο ενώ η BerkleyDB χρησιμοποιεί χώρο αποθήκευσης δεδομένων παραγγελίας κλειδιού/τιμής.

3.3.4. Βάση δεδομένων LevelDB από την πλευρά του πελάτη

Η LevelDB είναι μια γρήγορη βιβλιοθήκη αποθήκευσης κλειδιών/τιμών γραμμένη στην Google που παρέχει μια ταξινομημένη αντιστοίχιση από κλειδιά συμβολοσειρών σε τιμές συμβολοσειρών. Τα αποθηκευμένα δεδομένα ταξινομούνται κατά κλειδί και

παρέχουν μια ταξινομημένη αντιστοίχιση από τα κλειδιά συμβολοσειράς στις τιμές συμβολοσειρών (Dean, 2011).

3.3.5. Βάση δεδομένων WebSQL από την πλευρά του πελάτη

Το W3C(2010) έγραψε ότι το API της βάσης δεδομένων WebSQL είναι εκτός ενεργής συντήρησης. Ως αιτία αναφέρθηκε η έλλειψη ανεξάρτητων εφαρμογών επειδή το μεγαλύτερο μέρος του προγράμματος περιήγησης βασιζόταν στην SQLite ως την υποκείμενη βάση δεδομένων. Η βάση δεδομένων WebSQL έφερε την υλοποίηση πραγματικής σχεσιακής βάσης δεδομένων στα προγράμματα περιήγησης κάτι που σημαίνει ότι τα δεδομένα θα μπορούσαν να αποθηκευτούν με πολύ δομημένο τρόπο.

Όπως υποδηλώνει το όνομα, η βάση δεδομένων Web SQL Database[6] επιτρέπει την αποθήκευση και διαχείριση επίμονων (μακράς διάρκειας) σχεσιακών δεδομένων σε σχεσιακές βάσεις δεδομένων από την πλευρά του πελάτη μέσω της χρήσης JavaScript API και μιας παραλλαγής SQL. Για προγραμματιστές που είναι ήδη εξοικειωμένοι με τις σχεσιακές βάσεις δεδομένων του διακομιστή και τις δηλώσεις SQL, η χρήση της βάσης δεδομένων Web SQL μπορεί να φαίνεται σαν μία λογική επιλογή. Η μεγαλύτερη διαφορά μεταξύ των σχεσιακών βάσεων δεδομένων του πελάτη και του διακομιστή είναι ότι στις βάσεις δεδομένων του πελάτη ο χώρος αποθήκευσης είναι μάλλον περιορισμένος (προεπιλογή στα 5 MB ανά προέλευση).

Η Web SQL παρουσιάζει πολλά πλεονεκτήματα σε έναντι της αποθήκευσης Ιστού:

1. πιο σύνθετους τύπους δεδομένων,
2. πιο σύνθετα ερωτήματα (π.χ. συνδέσεις),
3. υποστήριξη για συναλλαγές και επανακλήσεις και
4. API σύγχρονων και ασύγχρονων βάσεων δεδομένων.

Υποστηρίζει ακόμη και προετοιμασμένες δηλώσεις για την αποφυγή εγχύσεων SQL. Όλα αυτά, ωστόσο επιφέρουν το κόστος ενός πιο περίπλοκου API.

Η **λίστα** που ακολουθεί περιγράφει ένα παράδειγμα του τρόπου χρήσης των API. Αρχικά, ορίζονται οι μεταβλητές και τα δεδομένα (γραμμές 1-8). Εν συνεχεία, δημιουργείται/ανοίγει μια βάση δεδομένων (γραμμές 10-12) και μέσα σε μια μεμονωμένη συναλλαγή δημιουργείται ένας πίνακας ενώ τα αρχικά δεδομένα

[6] Hickson, I. (eds.). *Web SQL Database*. W3C Working Group Note, November 2010. http://www.w3.org/TR/webdatabase/.

(συμπεριλαμβανομένου του JSON ως συμβολοσειράς) αποθηκεύονται στη βάση δεδομένων (γραμμές 14-23). Τέλος, τα δεδομένα ανακτώνται από τη βάση δεδομένων και μετατρέπονται ξανά σε JSON (γραμμές 25-33). Όπως μπορεί να φανεί, η ποσότητα του κώδικα είναι πολύ μεγαλύτερη και πολύπλοκη σε σύγκριση με το Web Storage, αλλά συνοδεύεται από πρόσθετη ευελιξία.

```
1: // Define variables
2: var myDatabase = null;
3: var version = "1";
4: // Define data
5: var myData = {
6: " code" : " T-111.5502" ,
7: " name" : " Seminar on Media Technology B P"
8: };
9: ...
10: // Create/open database
11: myDatabase = openDatabase( "myItemDB", version,
12: " My Item Database", 2 * 1024 * 1024 );
13: ...
14: // Start transaction
15: myDatabase.transaction( function( dbTx ) {
16: // Create table
17: dbTx.executeSql( "CREATE TABLE IF NOT EXISTS
18: myItems ( key TEXT UNIQUE, value TEXT )" );
19: // Store data
20: dbTx.executeSql( "INSERT INTO myItems VALUES
21: ( '1', '" + JSON.stringify( myData, null, " \t" )
22: + "' )" );
23: });
24: ...
25: // Start transaction
26: myDatabase.transaction( function( dbTx ) {
27: // Retrieve data
28: dbTx.executeSql( "SELECT * FROM myItems
29: WHERE key = '1'", [], function( exeTx, exeRS ) {
30: var myResult =
31: JSON.parse( exeRS.rows.item( 0 ).value );
32: });
33: });
```

Λίστα: Παράδειγμα σύνταξης της βάσης δεδομένων Web SQL: πρώτα δημιουργείται/ανοίγει μια βάση δεδομένων, μετά την οποία αποθηκεύονται και ανακτώνται τα δεδομένα.

Από τις 18 Νοεμβρίου 2010, οι εργασίες προδιαγραφής έχουν σταματήσει και η προδιαγραφή δεν βρίσκεται πλέον σε ενεργή συντήρηση. Ο λόγος πίσω από αυτήν την απόφαση ήταν ότι όλοι οι ενδιαφερόμενοι υλοποιητές έχουν χρησιμοποιήσει το ίδιο backend SQL, δηλαδή SQLite, και επομένως η διαδικασία τυποποίησης δεν μπορεί να προχωρήσει περαιτέρω. Επομένως, η χρήση του API δεν συνιστάται πλέον ενώ συνιστάται στους προγραμματιστές να χρησιμοποιούν το Indexed Database API (Mehta et al., 2012) για την επερχόμενη ανάπτυξη εφαρμογών Ιστού. Στην τρέχουσα κατάστασή του, ωστόσο, το API παρέχει μια βιώσιμη επιλογή για την αποθήκευση δεδομένων σε κινητές συσκευές, όπως φαίνεται στον **Πίνακα**.

Chrome	Firefox	IE	Opera	Safari	Android	iOS Safari
4.0+	—	—	10.5+	3.1 +	2.1+	3.2+

Πίνακας. Υποστήριξη προγράμματος περιήγησης για βάση δεδομένων Web SQL.

Υπάρχουν πολλές ανησυχίες σχετικά με το απόρρητο και την ασφάλεια που σχετίζονται με τους προαναφερθέντες μηχανισμούς αποθήκευσης από την πλευρά του πελάτη HTML5 που πρέπει να γνωρίζουν τόσο οι προγραμματιστές όσο και οι τελικοί χρήστες.

Όσον αφορά το απόρρητο των τελικών χρηστών, η μεγαλύτερη απειλή είναι η παρακολούθηση των χρηστών. Στην παρακολούθηση χρηστών, ένας διαφημιζόμενος τρίτου μέρους θα μπορούσε να χρησιμοποιήσει την περιοχή αποθήκευσης στην πλευρά του πελάτη για να παρακολουθεί έναν χρήστη σε πολλές περιόδους σύνδεσης, δημιουργώντας ένα προφίλ των ενδιαφερόντων του χρήστη για να επιτρέψει τη διαφήμιση υψηλής στόχευσης. Παρόλο που υπάρχει ένας αριθμός τεχνικών που μπορούν να χρησιμοποιηθούν για τον μετριασμό του κινδύνου παρακολούθησης των χρηστών (π.χ. λήξης αποθηκευμένων δεδομένων), ωστόσο η παρακολούθηση δεν μπλοκάρεται εντελώς (Mehta et al., 2012). Άλλες ανησυχίες που σχετίζονται με το απόρρητο είναι η χρήση των cookies HTTP ως πλεονάζον αντίγραφο ασφαλείας (ανάσταση cookie-cookie resurrection) και αντίστροφα, καθώς και η δυνατότητα

αποθήκευσης ευαίσθητων δεδομένων (π.χ. συναντήσεις ημερολογίου και αρχεία υγείας) είτε σε ιδιωτικούς ή δημόσιους πελάτες ιστού (Mehta et al., 2012).

Όλοι οι μηχανισμοί αποθήκευσης από την πλευρά του πελάτη HTML5 έχουν αυστηρή πολιτική ίδιας προέλευσης[7], που σημαίνει ότι διαφορετικοί ιστότοποι (προέλευσης, για την ακρίβεια) δεν μπορούν να έχουν πρόσβαση σε δεδομένα που δημιουργούνται από άλλο ιστότοπο (προέλευση). Ωστόσο, αυτό δεν σημαίνει ότι τα δεδομένα που είναι αποθηκευμένα στον χώρο αποθήκευσης από την πλευρά του πελάτη πρέπει να είναι αξιόπιστα, καθώς ένας κακόβουλος χρήστης μπορεί να έχει πρόσβαση στα δεδομένα (π.χ. χρησιμοποιώντας τα εργαλεία προγραμματιστών του Chrome) και να τα χειρίζεται όσο θέλει (Kessin, Z. (2011). Άλλες ανησυχίες που σχετίζονται με την ασφάλεια είναι οι επιθέσεις πλαστογράφησης DNS (που μπορούν να αποφευχθούν με χρήση TLS και πιστοποιητικών). Επιθέσεις μεταξύ καταλόγων, δηλαδή η χρήση μηχανισμών αποθήκευσης από την πλευρά του πελάτη σε ιστότοπους κοινής προέλευσης (π.χ. http://xxxxxxx.blogspot.com)· τόσο επιθέσεις SQL injection (καλό είναι να χρησιμοποιούνται πάντα προετοιμασμένες δηλώσεις) όσο και επιθέσεις JavaScript injection (XSS). και, φυσικά, ο κίνδυνος εφαρμογών κακής ποιότητας που θα μπορούσαν ενδεχομένως να οδηγήσουν σε πλαστογράφηση πληροφοριών.

Παρουσιάζονται ερευνητικά έργα που με τον ένα ή τον άλλο τρόπο χρησιμοποιούν πρόσφατους μηχανισμούς αποθήκευσης από την πλευρά του πελάτη με το ενδιαφέρον να επικεντρώνεται ειδικά σε δύο ερευνητικά έργα, το Sync Kit (Benson et al., 2010) και το Silo (Mickens, 2010).

Το Sync Kit είναι μια εργαλειοθήκη πελάτη/διακομιστή για τη βελτίωση της απόδοσης ιστοτόπων με ένταση δεδομένων. Χρησιμοποιεί JavaScript και (Google) Gears στην πλευρά του πελάτη, πράγμα που σημαίνει ότι η προσέγγιση απαιτεί την εγκατάσταση της προσθήκης Gears για να λειτουργήσει. Τα Gears θα μπορούσαν, ωστόσο, να αντικατασταθούν με έναν μηχανισμό αποθήκευσης από την πλευρά του πελάτη που καθορίζεται από το W3C για να αποφευχθεί η απαίτηση. Από την πλευρά του διακομιστή, απαιτείται διακομιστής web με επίγνωση του κιτ συγχρονισμού. Στην περίπτωσή τους, το Sync Kit υλοποιήθηκε ως μια συλλογή σεναρίων για το πλαίσιο web Django10.

[7] Barth, A. (eds.). *The Web Origin Concept*. IETF RFC 6454 (Standards Track), December 2011. http://www.ietf.org/rfc/rfc6454.txt

Η ιδέα πίσω από το Sync Kit είναι η χρήση των Gears για την αποθήκευση (cache) προτύπων ιστοσελίδων στην πλευρά του πελάτη. Τα πρότυπα περιλαμβάνουν βιβλιοθήκη JavaScript και ορισμούς τελικών σημείων δεδομένων για πρόσβαση σε δυναμικά περιεχόμενα που βρίσκονται στον διακομιστή. Τα τελικά σημεία δεδομένων, από την άλλη πλευρά, αποθηκεύουν προσωρινά αντικείμενα βάσης δεδομένων στα Gears και διατηρούν τα αποθηκευμένα δεδομένα στην κρυφή μνήμη συνεπή με τη βάση δεδομένων υποστήριξης. Όταν ένα πρόγραμμα περιήγησης ζητά μια ιστοσελίδα για πρώτη φορά, το πρότυπο επιστρέφεται ως απόκριση και αποθηκεύεται στον χώρο αποθήκευσης από την πλευρά του πελάτη. Στη συνέχεια, ζητούνται νέα δεδομένα μέσω των τελικών σημείων δεδομένων του προτύπου. Τέλος, τα δεδομένα που ανακτώνται προστίθενται στο πρότυπο, αποθηκεύονται προσωρινά στην αποθήκευση από την πλευρά του πελάτη και το αποτέλεσμα εμφανίζεται στον χρήστη.

Σύμφωνα με τους (Benson et al., 2010), η λύση που παρουσιάστηκε στην εργασία τους μειώνει το φόρτο του διακομιστή κατά τέσσερις φορές και τη μεταφορά δεδομένων έως και 5% σε σύγκριση με την παραδοσιακή προσέγγιση, όταν τα ποσοστά επίσκεψης στην κρυφή μνήμη είναι υψηλά.

Το Silo (Mickens, 2010) είναι ένα πλαίσιο για την ανάπτυξη εφαρμογών ιστού γρήγορης φόρτωσης. Χρησιμοποιεί JavaScript και Web Storage στην πλευρά του πελάτη (και οι δύο τυπικές τεχνολογίες web), που σημαίνει ότι η προσέγγιση λειτουργεί σε όλα τα σύγχρονα προγράμματα περιήγησης ιστού. Από την πλευρά του διακομιστή, απαιτείται διακομιστής ιστού Siloaware.

Η ιδέα πίσω από το Silo είναι η χρήση του Web Storage για την αποθήκευση (cache) JavaScript και CSS τμημάτων του ιστότοπου (μέγεθος 2 kB) στην πλευρά του πελάτη. Όταν ένα πρόγραμμα περιήγησης ζητά μια ιστοσελίδα, ο διακομιστής επιστρέφει μια τροποποιημένη έκδοση της ιστοσελίδας που περιέχει ένα μικρό παράρτημα JavaScript με μια λίστα απαιτούμενων αναγνωριστικών τμημάτων. Στη συνέχεια, η προσθήκη ελέγχει τον χώρο αποθήκευσης στην πλευρά του πελάτη για διαθέσιμα τμήματα και ενημερώνει τον διακομιστή για τα τμήματα που λείπουν χρησιμοποιώντας τα αναγνωριστικά τους. Στη συνέχεια, ο διακομιστής απαντά με τα ανεπεξέργαστα δεδομένα για τα τμήματα που λείπουν. Τέλος, η ιστοσελίδα ανακατασκευάζεται στην αρχική της μορφή στον πελάτη χρησιμοποιώντας το

πρόσθετο JavaScript. Επιπλέον, το δεύτερο στάδιο μπορεί να εξαλειφθεί εντελώς με τη χρήση cookies HTTP στο αρχικό αίτημα HTTP για την αποστολή των ήδη διαθέσιμων αναγνωριστικών τμημάτων. Το Silo αξιολογήθηκε με πολλούς ιστότοπους, όπως είναι το CNN, το Twitter και η Wikipedia και διαπιστώθηκε ότι είναι σε θέση να μειώσει τους χρόνους φόρτωσης ιστοσελίδων κατά 20-80% για ιστοσελίδες με μεγάλες ποσότητες JavaScript και CSS.

Οι μηχανισμοί αποθήκευσης από την πλευρά του πελάτη έχουν επίσης χρησιμοποιηθεί για την επίλυση άλλων σεναρίων χρήσης. Οι (Cannon & Wohlstadter, 2010) εφάρμοσαν ένα αυτοματοποιημένο πλαίσιο επιμονής αντικειμένων, το οποίο μειώνει την ποσότητα εργασίας που απαιτείται από τους προγραμματιστές για την υποστήριξη εφαρμογών ιστού εκτός σύνδεσης. Οι (Ijtihadie et al., 2010) παρουσίασαν ένα πρωτότυπο εφαρμογής Ιστού εκτός σύνδεσης για φορητές συσκευές για συγχρονισμό κουίζ για δραστηριότητες ηλεκτρονικής μάθησης, το οποίο βασίστηκε σε Εφαρμογές Ιστού και Αποθήκευση Ιστού εκτός σύνδεσης.

3.4. IndexedDB

Για την αποθήκευση μεγαλύτερων ποσοτήτων δεδομένων, η τοπική αποθήκευση και η αποθήκευση περιόδου λειτουργίας δεν είναι τόσο πρακτικές. Ανά κλειδί μπορεί να αποθηκευτεί μόνο μία τιμή. Οι προγραμματιστές Ιστού θα πρέπει να προσέξουν ιδιαίτερα τους τρόπους δομής των δεδομένων τους και ακόμη περισσότερο την υπολογιστική επιβάρυνση συνεπάγεται αυτή η δόμηση. Προκειμένου να διευκολυνθούν στο κομμάτι αυτό, έχει αναπτυχθεί μια άλλη μέθοδος αποθήκευσης δεδομένων και έχει προστεθεί στην HTML5: η IndexedDB (Mehta et al., 2015). Αυτή η API παρέχει έναν τρόπο αποθήκευσης συμβολοσειρών JSON με δομημένο τρόπο. Επιπλέον, καθιστά δυνατό τον ορισμό ευρετηρίων σε συγκεκριμένα στοιχεία στη βάση δεδομένων με σκοπό τη γρηγορότερη ανεύρεση. Το σύστημα δεν συμπεριφέρεται σαν μια παραδοσιακή σχεσιακή βάση δεδομένων αλλά μοιάζει πιο πολύ με τις τεχνικές NoSQL (Kimak & Ellman, 2015). Προφανώς αυτή η τεχνική αποθήκευσης αποθηκεύει δεδομένα επίμονα.

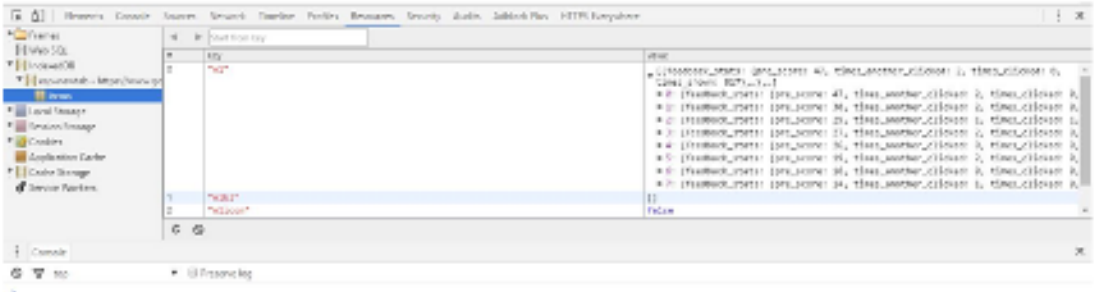
Υπάρχει επίσης ένα σύστημα βάσης δεδομένων που βασίζεται σε SQL: το Web SQL (Hickson, 2010). Ωστόσο, λόγω έλλειψης ανεξάρτητων εφαρμογών αυτής της τεχνικής η W3C αποφάσισε να μην διατηρήσει περαιτέρω αυτό το πρότυπο.

Η αποθήκευση δεδομένων σε μια βάση δεδομένων IndexedDB είναι πιο περίπλοκη από ό,τι στην τοπική αποθήκευση. Αρχικά πρέπει να ρυθμιστεί μια σύνδεση σε μια από τις τοπικές βάσεις δεδομένων που είναι αποθηκευμένες από το πρόγραμμα περιήγησης. Εάν η βάση δεδομένων δεν υπάρχει ακόμη, τότε τα προγράμματα περιήγησης θα δημιουργήσουν μία.

Όταν έχει γίνει μια σύνδεση, τα αντικείμενα μπορούν να αποθηκευτούν χρησιμοποιώντας συναλλαγές βάσης δεδομένων. Με τις συναλλαγές μπορούν να ζητηθούν αντικείμενα από τη βάση δεδομένων και στη συνέχεια να προστεθούν στοιχεία σε αυτά τα αντικείμενα. Ένα στοιχείο πρέπει να καθοριστεί ως συμβολοσειρά JSON.

Το IndexedDB χρησιμοποιεί τον χειρισμό σφαλμάτων ως προεπιλεγμένη συμπεριφορά. Επομένως, οι περισσότερες μέθοδοι δεν επιστρέφουν τις τιμές τους αμέσως. Αρχικά, όσο αφορά τους χειριστές συμβάντων πρέπει να προσδιορίζεται το τι πρέπει να κάνουν εάν μια λειτουργία επιτύχει ή πότε παρουσιάζει σφάλμα. Η ανάγνωση και η εγγραφή σε βάσεις δεδομένων IndexedDB μπορεί να γίνει μόνο μέσω JavaScript..

Όπως και με την τοπική αποθήκευση, οι βάσεις δεδομένων IndexedDB μπορούν να εμφανίζονται στα development kits του Google Chrome και του Mozilla Firefox. Ένα παράδειγμα αυτής της άποψης φαίνεται στην ακόλουθη **εικόνα**.



Εικόνα: Πόροι IndexedDB of google.nl που εμφανίζονται στο Google Chrome

3.4.1. Βάση δεδομένων IndexedDB από την πλευρά του πελάτη

Η IndexedDB της HTML5 είναι μια βάση δεδομένων από την πλευρά του πελάτη ενσωματωμένη στο πρόγραμμα περιήγησης ιστού του πελάτη. Η εφαρμογή χρησιμοποιεί τοπικά δεδομένα που είναι αποθηκευμένα σε ένα σύστημα πελάτη. Αποθηκεύει στην κρυφή μνήμη μεγάλα δεδομένα από το διακομιστή στην πλευρά του

πελάτη χρησιμοποιώντας JavaScript Object Stores, ισοδύναμα με πίνακες σε σχεσιακές βάσεις δεδομένων.

Μια εφαρμογή αποθηκεύει αντικείμενα JavaScript στο IndexedDB όταν η εφαρμογή είναι συνδεδεμένη στο Διαδίκτυο. Όταν η σύνδεση τερματιστεί για οποιονδήποτε λόγο, η εφαρμογή μπορεί να ανακτήσει δεδομένα από το IndexedDB και στη συνέχεια μπορεί να εκτελεστεί εκτός σύνδεσης Η εφαρμογή εκτελείται σαν να ήταν εφαρμογή της επιφάνειας εργασίας. Το παραπάνω είναι ιδιαίτερα χρήσιμο σε πελάτες κινητής τηλεφωνίας, καθώς μπορούν να χρησιμοποιήσουν την εφαρμογή ακόμα και αν η σύνδεση χαθεί λόγω κακού σήματος.

Οι βάσεις δεδομένων IndexedDB αποθηκεύουν ζεύγη κλειδιών/τιμών. Οι τιμές μπορεί να είναι σύνθετα δομημένα αντικείμενα και τα κλειδιά μπορεί να είναι ιδιότητες αυτών των αντικειμένων. Τα ευρετήρια χρησιμοποιούν την ιδιότητα των αντικειμένων για γρήγορη αναζήτηση και ταξινομημένη απαρίθμηση. Ένα κλειδί είναι μια τιμή δεδομένων με την οποία οι αποθηκευμένες τιμές οργανώνονται και ανακτώνται στο χώρο αποθήκευσης αντικειμένων.

Το IndexedDB είναι χτισμένο σε ένα μοντέλο βάσης δεδομένων συναλλαγών.

Όλα όσα γίνονται στο IndexedDB συμβαίνουν πάντα στο πλαίσιο μιας συναλλαγής. Μια συναλλαγή είναι ένα ατομικό και ανθεκτικό σύνολο λειτουργιών πρόσβασης και τροποποίησης δεδομένων σε μια συγκεκριμένη βάση δεδομένων. Είναι ο τρόπος με τον οποίο ένα πρόγραμμα περιήγησης αλληλεπιδρά με τα δεδομένα μιας βάσης δεδομένων. Οποιαδήποτε ανάγνωση ή αλλαγή δεδομένων στη βάση δεδομένων πρέπει να συμβαίνει σε μια συναλλαγή. Το IndexedDB API παρέχει πολλά αντικείμενα που αντιπροσωπεύουν ευρετήρια, πίνακες, δρομείς, αλλά καθένα από αυτά συνδέεται με μια συγκεκριμένη συναλλαγή. Δεν μπορεί να εκτελεστεί μια εντολή ή να ανοίξει ο κέρσορας εκτός μιας συναλλαγής. Οι συναλλαγές έχουν καθορισμένη διάρκεια ζωής, επομένως η απόπειρα χρήσης μιας συναλλαγής μετά την ολοκλήρωσή της δημιουργεί εξαιρέσεις (W3C, 2011). Το IndexedDB δεν χρησιμοποιεί SQL αλλά χρησιμοποιεί ερωτήματα σε ένα ευρετήριο που παράγει έναν δρομέα, ο οποίος με τη σειρά του χρησιμοποιείται για επανάληψη στο σύνολο αποτελεσμάτων. Το ευρετήριο είναι μια δομή δεδομένων (ένας τρόπος αποθήκευσης και οργάνωσης δεδομένων) που βελτιώνει την ανάκτηση δεδομένων από τη βάση δεδομένων. Η δομή μιας βάσης δεδομένων IndexedDB μπορεί να τροποποιηθεί μόνο

κατά τη διάρκεια μιας συναλλαγής αλλαγής έκδοσης. Αυτό σημαίνει ότι η μόνη φορά που μπορούν να δημιουργηθούν ή να αφαιρεθούν ObjectStores ή ευρετήρια είναι κατά τη διάρκεια της συναλλαγής versionchange. Βασικά, η IndexedDB API δημιουργεί αυτόματα μια συναλλαγή αλλαγής έκδοσης κάθε φορά που ανοίγει μια βάση δεδομένων μέσω της ανοιχτής μεθόδου και προκύπτει μία από τις ακόλουθες δύο συνθήκες:

- Η ζητούμενη βάση δεδομένων δεν υπάρχει.
- Ο αριθμός έκδοσης της ζητούμενης βάσης δεδομένων είναι μεγαλύτερος από τον αριθμό έκδοσης της βάσης δεδομένων στον υπολογιστή-πελάτη.

3.4.2. Νέες υλοποιήσεις ανοιχτών βάσεων δεδομένων

Επί του παρόντος, το Chrome εξακολουθεί να εφαρμόζει την παλιά προδιαγραφή και όχι τη νέα. Παρομοίως, εξακολουθεί να έχει την ιδιότητα webkitIndexedDB με πρόθεμα, ακόμη και αν υπάρχει το indexedDB χωρίς πρόθεμα.

Ορισμένα νέα έργα που βασίζονται σε IndexedDB είναι τα εξής:

- PouchDB - Μια υλοποίηση του CouchDB πάνω από το IndexedDB. Μία από τις προϋποθέσειςτου είναι είναι να προσφέρει τις ίδιες αποκεντρωμένες δυνατότητες συγχρονισμού (master-to-master) του CouchDB στο πρόγραμμα περιήγησης.
- BrowserCouch – Πρόκειται για ένα παρόμοιο έργο αλλά χρησιμοποιεί WebSQL ως χώρο αποθήκευσης προγράμματος περιήγησης. Το Html5sql είναι μια ελαφριά λειτουργική μονάδα JavaScript (βιβλιοθήκη jQuery) που απλοποιεί την εργασία με το IndexedDB. Η κύρια λειτουργία του είναι να παρέχει μια δομή για μία «ακολουθιακή» επεξεργασία των δηλώσεων SQL μέσα σε μια μεμονωμένη συναλλαγή. Αυτό από μόνο του απλοποιεί σημαντικά την αλληλεπίδραση με τη βάση δεδομένων, ωστόσο δεν σταματά εκεί. Πολλά άλλα μικρότερα χαρακτηριστικά έχουν συμπεριληφθεί για να κάνουν τα πράγματα ευκολότερα, πιο φυσικά και πιο βολικά για τους προγραμματιστές.

3.4.3. Διαφορές μεταξύ NoSQL και βάσης δεδομένων SQL

Η κύρια διαφορά μεταξύ NoSQL και βάσης δεδομένων SQL είναι ο τρόπος αποθήκευσης των δεδομένων. Το NoSQL χρησιμοποιεί κλειδί/τιμή ως κύρια

αποθήκευση. Από την άλλη πλευρά, η SQL είναι μια σχεσιακή βάση δεδομένων, που σημαίνει ότι χρησιμοποιεί σχέσεις (που ονομάζονται πίνακες). Οι βάσεις δεδομένων διαφέρουν ως προς την επεκτασιμότητα και την απόδοση. Η βάση δεδομένων NoSQL έχει πλεονεκτήματα έναντι της βάσης δεδομένων SQL επειδή επιτρέπει την κλιμάκωση μιας εφαρμογής σε νέα επίπεδα. Οι νέες υπηρεσίες δεδομένων απαιτούν κλιμακούμενες δομές, οι οποίες μπορούν να λειτουργήσουν στο cloud. Συγκριτικά, η βάση δεδομένων NoSQL δεν χρειάζεται διαχειριστή βάσης δεδομένων ή περίπλοκα ερωτήματα SQL και εξακολουθεί να θεωρείται ταχύτερη στη διαχείριση μεγάλου όγκου δεδομένων. Ωστόσο, το NoSQL δεν υποστηρίζει συνδέσεις SQL και οι σχέσεις μεταξύ των πινάκων πρέπει να προγραμματίζονται με μη αυτόματο τρόπο. Οι διαφορές μεταξύ σχεσιακών και IndexedDB βάσεων δεδομένων έγκεινται στην αποθήκευση των δεδομένων. Οι σχεσιακές βάσεις δεδομένων αποθηκεύουν πίνακες με γραμμές και στήλες τύπων δεδομένων. Το IndexedDB απαιτεί τη δημιουργία ενός χώρου αποθήκευσης αντικειμένων για τον τύπο δεδομένων και την αποθήκευση αντικειμένων JavaScript σε αυτόν τον χώρο αποθήκευσης. Κάθε αντικείμενο μπορεί να έχει συλλογή από ευρετήρια που το καθιστούν πιο γρήγορο στην αναζήτηση. Η IndexedDB δεν υποστηρίζει συνδέσεις, όπου αυτές υποστηρίζονται από τη σχεσιακή βάση δεδομένων. Η σύγκριση των αποτελεσμάτων του ερωτήματος χρησιμοποιώντας συνδέσεις δείχνει ότι το IndexedDB εκτελεί το ερώτημα και αποδίδει τα δεδομένα πιο γρήγορα.

Το IndexedDB είναι πολύ πιο περίπλοκο, καθώς όλος ο κώδικας πρέπει να γίνεται χειροκίνητα σε JavaScript που παρέχεται εγγενώς από την SQL. Η IndexedDB μπορεί να χωρίσει τον πίνακα σε αποσπάσματα μικρών τμημάτων και χρησιμοποιώντας το setTimeout, αντί για βρόχο, τα δεδομένα εισάγονται πιο γρήγορα στη βάση δεδομένων.

3.5. Μέθοδος παρακολούθησης

Λόγω των μέτρων απορρήτου που λαμβάνονται στα προγράμματα περιήγησης για την αποτροπή της παρακολούθησης χρηστών μέσω Local Storage και IndexedDB, ένα πολύ σημαντικό ζήτημα είναι το πώς μπορεί να γίνει η παρακολούθηση χρηστών.

Η πρόσβαση στους πόρους τοπικής αποθήκευσης και IndexedDB είναι δυνατή μόνο από την ακριβή προέλευση που αποθήκευσε τις πληροφορίες. Επομένως, οι

πληροφορίες που αποθηκεύονται από έναν ιστότοπο δεν μπορούν να διαβαστούν από έναν άλλο.

Η εφαρμογή της Πολιτικής Ίδιας Προέλευσης προβλέπει ότι εάν οι υπηρεσίες παρακολούθησης θέλουν να παρακολουθήσουν έναν χρήστη σε διαφορετικούς ιστότοπους, πρέπει να ενσωματώσουν ένα πλαίσιο σε όλους τους ιστότοπους. Εάν το πλαίσιο χρησιμοποιεί την προέλευση της συγκεκριμένης υπηρεσίας παρακολούθησης, οι πόροι τοπικής αποθήκευσης και IndexedDB μπορούν να προσπελαστούν από αυτό το πλαίσιο.

Μια υπηρεσία παρακολούθησης συνήθως θέλει επίσης να γνωρίζει τις λεπτομέρειες του ιστότοπου που επισκέπτεται ο χρήστης. Επομένως, το ενσωματωμένο πλαίσιο πρέπει να επικοινωνεί με τον ιστότοπο-γονιό για να λάβει αυτές τις λεπτομέρειες. Οι λεπτομέρειες μπορούν στη συνέχεια να συνδεθούν με ένα αναγνωριστικό χρήστη μέσω του πλαισίου και στη συνέχεια όλα τα δεδομένα μπορούν, για παράδειγμα, να προωθηθούν στην υπηρεσία παρακολούθησης.

Η επικοινωνία μεταξύ των πλαισίων μπορεί δυνητικά να γίνει με διάφορους τρόπους:

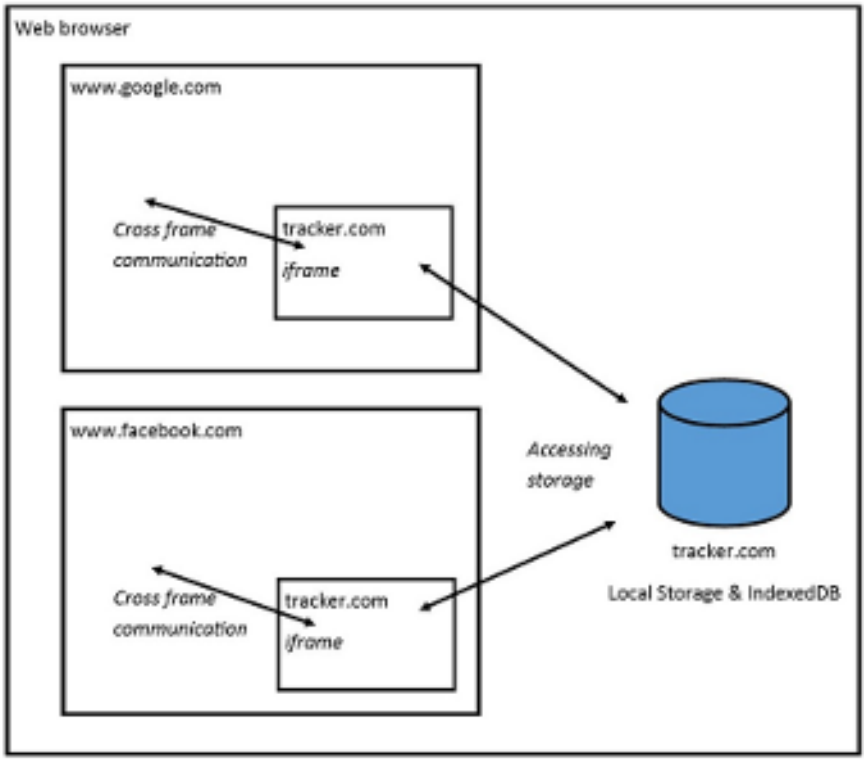
• Ο πιο απλός τρόπος είναι το γεγονός ότι οι λεπτομέρειες του ιστότοπου αναφέρονται στη διεύθυνση URL πηγής του πλαισίου. Μπορούν για παράδειγμα, να χρησιμοποιηθούν παράμετροι GET.

•Τα πλαίσια μπορεί να έχουν άμεση πρόσβαση στο DOM των γονικών πλαισίων. Με αυτόν τον τρόπο τα στοιχεία του ιστότοπου μπορούν απλά να ζητηθούν μέσω JavaScript.

• Υπάρχει επίσης ένα ειδικό σύστημα επικοινωνίας διαθέσιμο σε προγράμματα περιήγησης για την επικοινωνία μεταξύ πλαισίων: τα Μηνύματα Ιστού. Η κλήση της συνάρτησης JavaScript postMessage() ενός άλλου πλαισίου καθιστά δυνατή την αποστολή μηνυμάτων στα πλαίσια. Το πλαίσιο λήψης πρέπει να ακούει ρητά για μηνύματα, επομένως απαιτείται συνεργασία τόσο της πλευράς αποστολής όσο και της πλευράς λήψης. Σύμφωνα με την προδιαγραφή δεν υπάρχουν ενσωματωμένοι περιορισμοί σχετικά με αυτήν την επικοινωνία, επομένως αυτή η ίδια η Πολιτική Ίδιας Προέλευσης μπορεί να παρακαμφθεί και τα δύο μέρη εργάζονται χωριστά.

Δεν μπορούν να χρησιμοποιηθούν στην πράξη όλες οι επιλογές για επικοινωνία μεταξύ πλαισίων λόγω των μέτρων προστασίας στα προγράμματα περιήγησης ιστού.

Η γενική ιδέα ώστε να καταστεί δυνατή η παρακολούθηση χρηστών μέσω Τοπικής αποθήκευσης και IndexedDB χρησιμοποιώντας ενσωματωμένα πλαίσια απεικονίζεται στην παρακάτω **εικόνα**.



Εικόνα: Γραφική αναπαράσταση της παρακολούθησης χρηστών με χρήση Τοπικής αποθήκευσης και/ή IndexedDB

3.6. FileSystem

Το HTML5 Filesystem API δίνει στις εφαρμογές τη δυνατότητα να γράφουν και να αποθηκεύουν πραγματικά αρχεία σε JavaScript. Οι ιστότοποι έχουν πλέον τη δυνατότητα να αποθηκεύουν μεγάλες ποσότητες δυαδικών δεδομένων στο σύστημα ενός χρήστη. Είναι βέβαια σημαντικό οι εφαρμογές να μην κάνουν κατάχρηση ενός τέτοιου πλεονεκτήματος, όπως για παράδειγμα, να μην καταναλώνουν μεγάλες ποσότητες χώρου στο δίσκο χωρίς τη γνώση ή τη συγκατάθεση του χρήστη. Το τελευταίο πράγμα που θέλουν οι χρήστες είναι να έχουν αποθηκευμένα 20 GB δεδομένων στο σύστημά τους απλώς επισκεπτόμενοι μια διεύθυνση URL.

Τη στιγμή της σύνταξης, το Chrome διαθέτει μια περιορισμένη σελίδα ρυθμίσεων διεπαφής χρήστη για να διαχειρίζονται οι χρήστες τον χώρο αποθήκευσης για

εφαρμογές που αποθηκεύουν δεδομένα για λογαριασμό τους. Είναι προσβάσιμο μέσω Preferences→ Under the Hood→ All Cookies and Site Data (ή ανοίγοντας τα cookies chrome://settings/). Οι χρήστες μπορούν να διαγράψουν δεδομένα μόνο από αυτό το μενού. Ως αποτέλεσμα αυτής της περιορισμένης διεπαφής χρήστη, οι λειτουργίες εγγραφής (όπως η δημιουργία ενός φακέλου και η εγγραφή σε ένα αρχείο) απαιτούν από μια εφαρμογή να ζητά το εκτιμώμενο μέγεθος, σε byte, που αναμένεται να χρησιμοποιήσει. Η ίδια πρακτική ισχύει και για άλλα API αποθήκευσης εκτός σύνδεσης, όπως το WebSQL DB, όπου κάποιος ανοίγει μια βάση δεδομένων με συγκεκριμένο μέγεθος:

```
var db = window.openDatabase(
 'MyDB', // dbName
 '1.0', // version
 'test database', // description
 2 * 1024 * 1024, // estimatedSize in bytes (2MB)
 function(db) {} // optional creationCallback
);
```

Μια κανονική εφαρμογή Ιστού μπορεί να ζητήσει χώρο αποθήκευσης σε δύο κατηγορίες: προσωρινά ή μόνιμα. Εκτός από αυτούς τους τύπους, οι επεκτάσεις Chrome και οι φιλοξενούμενες εφαρμογές ιστού που αναφέρονται στο Chrome Web Store έχουν μια τρίτη επιλογή: απεριόριστο αποθηκευτικό χώρο.

3.6.1. Προσωρινή αποθήκευση

Η προσωρινή αποθήκευση είναι ευκολότερο να αποκτηθεί. Στην πραγματικότητα, δεν χρειάζεται καν να ζητηθεί. Από προεπιλογή, δίνεται στις αρχές μια μικρή ποσότητα προσωρινής αποθήκευσης, που σημαίνει ότι η προσωρινή αποθήκευση μπορεί να αποκτηθεί χωρίς ειδικές άδειες ή ότι το πρόγραμμα περιήγησης ζητά από τον χρήστη να προβεί σε κάποια ενέργεια. Η προσωρινή αποθήκευση είναι ιδανική για πράγματα όπως η κρυφή μνήμη.

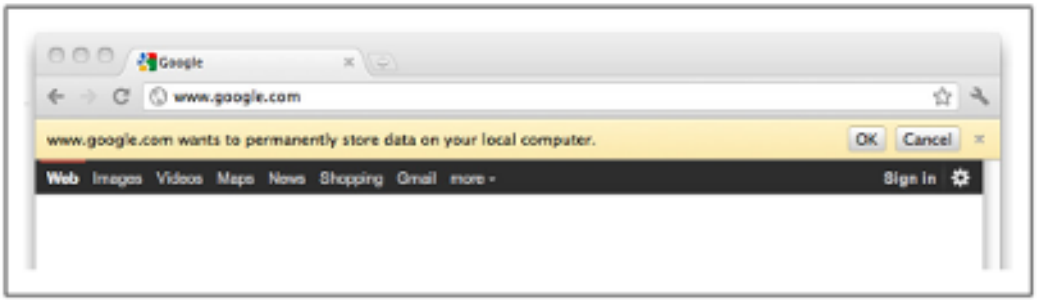
Στο Google Chrome 13, το σύστημα αρχείων HTML5 και το WebSQL DB μοιράζονται μια δεξαμενή χώρου στο δίσκο που οι ιστότοποι μπορούν να

καταναλώσουν συλλογικά. Ένας μεμονωμένος ιστότοπος μπορεί να καταναλώσει έως και το 20% της δεξαμενής. Καθώς η χρήση της προσωρινής δεξαμενής πλησιάζει το όριο για το σύνολο (1 GB), τα δεδομένα που χρησιμοποιήθηκαν λιγότερο πρόσφατα θα ανακτηθούν. Τελικά, η προσωρινή μνήμη εφαρμογών και η IndexedDB θα μοιράζονται επίσης αυτήν την προσωρινή δεξαμενή. Ένα τέτοιο ενοποιημένο σύστημα ποσοστώσεων σημαίνει επίσης ότι δεν υπάρχει πλέον το όριο 5 MB που επιβάλλεται στο WebSQL DB.

3.6.2. Μόνιμη αποθήκευση

Στη μόνιμη αποθήκευση τα δεδομένα που αποθηκεύονται με αυτήν την επιλογή είναι διαθέσιμα σε επόμενες προσβάσεις στο ίδιο σύστημα αρχείων. Πρέπει να ληφθεί υπόψη ωστόσο, ότι ακόμη και τα μόνιμα δεδομένα μπορούν να διαγραφούν με μη αυτόματο τρόπο από τον χρήστη (είτε μέσω μιας σελίδας ρυθμίσεων προγράμματος περιήγησης είτε μέσω άμεσων λειτουργιών του συστήματος αρχείων στο λειτουργικό σύστημα). Επομένως, τα δεδομένα που αποθηκεύονται δεν είναι ποτέ 100% σίγουρο ότι θα υπάρχουν.

Μια βασική διαφορά από την προσωρινή αποθήκευση είναι ότι το πρόγραμμα περιήγησης ζητά άδεια από τον χρήστη πριν εκχωρήσει μόνιμο χώρο αποθήκευσης. Στο Chrome, αυτό εμφανίζεται ως γραμμή πληροφοριών (δείτε Εικόνα 2-1).



Εικόνα . Το πρόγραμμα περιήγησης προτρέπει τον χρήστη όταν ζητηθεί μόνιμος χώρος αποθήκευσης

Επειδή η παρέμβαση χρήστη εμπλέκεται σε αυτήν την επιλογή αποθήκευσης, οι εφαρμογές έχουν μηδενικό μόνιμο όριο από προεπιλογή. Οποιεσδήποτε προσπάθειες αποθήκευσης περισσότερων δεδομένων από το όριο που έχει χορηγηθεί πρόκειται να αποτύχουν με QUOTA_EXCEEDED_ERR.

Ιδιότητες της μόνιμης αποθήκευσης:

• Το πρόγραμμα περιήγησης ρωτά τον χρήστη εάν θα ζητηθεί επιπλέον χώρος.

• Οι εφαρμογές έχουν μηδενικό όριο από προεπιλογή.

• Εάν χρειάζεται περισσότερος χώρος αποθήκευσης, μπορεί να ζητηθεί. Δεν υπάρχει δεξαμενή αποθήκευσης σταθερού μεγέθους.

• Τα δεδομένα είναι εγγυημένα διαθέσιμα σε επόμενες προσβάσεις.

3.6.3. Απεριόριστος χώρος αποθήκευσης

Ο απεριόριστος χώρος αποθήκευσης είναι μια μοναδική επιλογή για τις επεκτάσεις και τις εφαρμογές του Chrome που παρατίθενται στο Chrome Web Store (είτε είναι εγκατεστημένο είτε όχι). Χρησιμοποιώντας την άδεια unlimitedStorage στο αρχείο .manifest, μπορεί κανείς να παρακάμψει τους περιορισμούς της προσωρινής και μόνιμης αποθήκευσης. Ο απεριόριστος χώρος αποθήκευσης πρέπει να θεωρηθεί ως μόνιμος χώρος αποθήκευσης, αλλά χωρίς προτροπή χρήστη και μέγιστο όριο.

Ιδιότητες της unlimitedStorage:

• Αποκλειστικά για τις εφαρμογές και τις επεκτάσεις του Chrome.

• Χορηγείται απεριόριστο όριο χωρίς προτροπές χρήστη (εκτός από τη στιγμή της εγκατάστασης).

• Δεν χρειάζεται να ζητηθεί περισσότερος αποθηκευτικός χώρο όταν χρειάζεται περισσότερος χώρος.

3.6.4. Διαχείριση Ποσοστώσεων

Το Chrome 13 πρόσθεσε ένα API διαχείρισης ορίων για να δώσει στις εφαρμογές ένα εργαλείο για να ζητούν, να διαχειρίζονται και, το πιο σημαντικό, να ρωτούν τον τρέχοντα χώρο αποθήκευσης που καταλαμβάνει η προέλευσή τους. Το API εκτίθεται ως νέο καθολικό αντικείμενο, webkitStorageInfo: window.webkitStorageInfo Το API quota έχει πρόθεμα επειδή δεν είναι ακόμη τυποποιημένο. Διαθέτει δύο μεθόδους:

queryUsageAndQuota (τύπος, opt_successCallback, opt_errorCallback);

τύπος

Ο τύπος χώρου αποθήκευσης για την επιστροφή της τρέχουσας χρήσης. Πιθανές τιμές είναι TEMPO RARY ή PERSISTENT.

opt_successCallback

Μια προαιρετική επανάκληση δύο παραμέτρων. Οι παράμετροι είναι ο τρέχων αριθμός byte που χρησιμοποιεί η εφαρμογή και το τρέχον όριο, επίσης σε byte.

opt_errorCallback

Προαιρετικό σφάλμα επανάκλησης. requestQuota (τύπος, μέγεθος, opt_successCallback, opt_errorCallback);

τύπος

Εάν η νέα/πρόσθετη αποθήκευση θα πρέπει να είναι μόνιμη ή προσωρινή. Πιθανές τιμές είναι Temporary ή Persistent.

opt_successCallback

Μια προαιρετική επανάκληση πέρασε το παραχωρημένο όριο σε byte.

opt_errorCallback

Προαιρετικό σφάλμα επανάκλησης.

3.6.5. Επιπλέον χώρος αποθήκευσης

Για να ζητηθεί νέος ή πρόσθετος χώρος αποθήκευσης, πρέπει να κληθεί η requestQuota() με τον τύπο του χώρου αποθήκευσης, το μέγεθος και μια επιτυχή επανάκληση. Το πρόγραμμα περιήγησης ζητά από τον χρήστη μια γραμμή αδειών όταν ζητείται ΜΟΝΙΜΟΣ αποθηκευτικός χώρος. Εάν το μέγεθος που μεταβιβάστηκε στην requestQuota() είναι μικρότερο από την τρέχουσα κατανομή της εφαρμογής, δεν εμφανίζεται κανένα μήνυμα. Η τρέχουσα ποσόστωση διατηρείται. Εάν η εφαρμογή ζητά επιπλέον χώρο (π.χ., το νέο μέγεθος είναι μεγαλύτερο από το υπάρχον όριο της εφαρμογής), θα ζητηθεί εκ νέου από τον χρήστη να αποδεχτεί αυτήν την αλλαγή. Εάν το αίτημα είναι για ΠΡΟΣΩΡΙΝΗ αποθήκευση, και πάλι, δεν θα εμφανιστεί κανένα μήνυμα προτροπής, αλλά άλλα δεδομένα ενδέχεται να εξαλειφθούν κατά την κρίση του προγράμματος περιήγησης.

Το ακόλουθο παράδειγμα ζητά 2 MB PERSISTENT αποθηκευτικού χώρου:

```
window.webkitStorageInfo.requestQuota(PERSISTENT, 2*1024*1024,
function(bytes) {
```

```
 console.log('Granted ' + bytes + ' bytes in persistent storage');

}, function(e) {

 console.log('Error', e);

});
```

3.6.6. Έλεγχος Τρέχουσας Χρήσης

Για να υποβληθεί ερώτημα για την τρέχουσα χρήση αποθηκευτικού χώρου και το όριο μιας εφαρμογής, πρέπει να κληθεί το queryUsageAnd Quota() με τον τύπο χώρου αποθήκευσης που είναι υπό έλεγχο και μια επιτυχή επανάκληση. Αυτή η μέθοδος επιστρέφει δύο πράγματα στην επανάκλησή, τον αριθμό των byte που χρησιμοποιούνται και το συνολικό όριο για τον εν λόγω τύπο αποθήκευσης.

Για παράδειγμα, εάν το example.com ήθελε να ελέγξει το ποσοστό της ΠΡΟΣΩΡΙΝΗΣ αποθήκευσης που χρησιμοποιεί, θα μπορούσε να εκτελεστεί ως εξής:

```
window.webkitStorageInfo.queryUsageAndQuota(TEMPORARY, function(usage,
quota) {

 console.log('Using: ' + (usage / quota) * 100 + '% of temporary storage');

}, function(e) {

 console.log('Error', e);

});
```

Η χρήση που αναφέρεται από το quota API ενδέχεται να μην ταιριάζει ακριβώς με το μέγεθος που ζητήθηκε χρησιμοποιώντας την requestQuota() ή το πραγματικό μέγεθος των αποθηκευμένων δεδομένων στο δίσκο. Η απόκλιση προέρχεται από κάθε τύπο αποθήκευσης που χρειάζεται κάποια επιπλέον byte για την αποθήκευση μεταδεδομένων. Ενδέχεται επίσης να υπάρχει κάποια χρονική καθυστέρηση έως ότου οι ενημερώσεις αντικατοπτρίζονται στο API ορίου.

3.6.7. Άνοιγμα συστήματος αρχείων

Μια εφαρμογή Ιστού αποκτά πρόσβαση στο σύστημα αρχείων HTML5 ζητώντας ένα αντικείμενο LocalFile System χρησιμοποιώντας μια καθολική μέθοδο,

window.requestFileSystem():window.requestFileSystem(τύπος,μέγεθος, successCallback, opt_errorCallback)

Οι παράμετροί του περιγράφονται παρακάτω:

τύπος

Στην περίπτωση που η αποθήκευση πρέπει να είναι μόνιμη. Πιθανές τιμές είναι TEMPORARY ή PERSISTENT. Τα δεδομένα που έχουν αποθηκευτεί με τη χρήση TEMPORARY μπορούν να αφαιρεθούν κατά την κρίση του προγράμματος περιήγησης (για παράδειγμα, εάν απαιτείται περισσότερος χώρος). Ο ΜΟΝΙΜΟΣ αποθηκευτικός χώρος δεν μπορεί να εκκαθαριστεί εκτός εάν εξουσιοδοτηθεί ρητά από τον χρήστη ή την εφαρμογή.

Μέγεθος

Είναι ένας δείκτης του πόσο αποθηκευτικό χώρο, σε byte, αναμένει να χρειαστεί η εφαρμογή.

successCallback

Μια συνάρτηση επανάκλησης που καλείται όταν ο παράγοντας χρήστη παρέχει με επιτυχία ένα σύστημα αρχείων. Το όρισμά του είναι ένα αντικείμενο FileSystem.

opt_error Callback

Μια προαιρετική συνάρτηση επανάκλησης που καλείται όταν παρουσιαστεί σφάλμα ή απορρίπτεται το αίτημα για ένα σύστημα αρχείων. Το όρισμά της είναι ένα αντικείμενο FileError.

Η κλήση της window.requestFileSystem() για πρώτη φορά δημιουργεί έναν νέο χώρο αποθήκευσης sandbox για την εφαρμογή και την προέλευση που το ζήτησε. Ένα σύστημα αρχείων περιορίζεται σε μία μόνο εφαρμογή και δεν μπορεί να έχει πρόσβαση στα αποθηκευμένα δεδομένα άλλης εφαρμογής. Αυτό σημαίνει επίσης ότι μια εφαρμογή δεν μπορεί να διαβάσει/εγγράψει αρχεία σε έναν αυθαίρετο φάκελο στον σκληρό δίσκο του χρήστη. Κάθε σύστημα αρχείων είναι απομονωμένο.

Παράδειγμα. Αίτημα προσωρινής αποθήκευσης συστήματος αρχείων

```
var onError = function(fs) {
```

```
 console.log('There was an error');
};
var onFS = function(fs) {
 console.log('Opened filesystem: ' + fs.name);
};
window.requestFileSystem(window.TEMPORARY, 5*1024*1024 /*5MB*/, onFs,
onError);
```

Εάν όλα πάνε καλά, η επιτυχής επανάκληση (onFS) καλείται οπότε μεταβιβάζεται ένα αντικείμενο FileSystem που περιέχει δύο ιδιότητες:

όνομα

Ένα μοναδικό όνομα για το σύστημα αρχείων, που εκχωρείται από το πρόγραμμα περιήγησης

ρίζα

Ένα DirectoryEntry μόνο για ανάγνωση που αντιπροσωπεύει τη ρίζα του συστήματος αρχείων

Το αντικείμενο FileSystem είναι η πύλη του χρήστη σε ολόκληρο το API. Από τη στιγμή που υπάρχει μία αναφορά, αξίζει αυτή να αποθηκευτεί στην προσωρινή μνήμη σε μια καθολική μεταβλητή ή ιδιότητα κλάσης. Θα χρησιμοποιηθεί παντού. Τα πράγματα γίνονται λίγο πιο περίπλοκα στην περίπτωση της μόνιμης αποθήκευσης με το σύστημα αρχείων. Οι αιτήσεις έχουν μηδενική επίμονη ποσόστωση από προεπιλογή. Ως αποτέλεσμα, πρέπει να ζητηθεί κάποιο μόνιμο όριο πριν ανοιχθεί το σύστημα αρχείων, το οποίο μπορεί να σημαίνει απλώς την αναδίπλωση της κλήσης στο window.requestFileSystem() στο requestQuota() επανάκληση.

Παράδειγμα. Ζητώντας ένα σύστημα αρχείων με μόνιμο χώρο αποθήκευσης

```
const SIZE = 5*1024*1024; /*5MB*/
const TYPE = PERSISTENT;
```

```
window.webkitStorageInfo.requestQuota(TYPE, SIZE, function(grantedBytes) {
 window.requestFileSystem(TYPE, grantedBytes, onFs, onError);
}, function(e) {
 console.log('Error', e);
});
```

Αφού ο χρήστης παραχωρήσει άδεια χρήσης μόνιμου αποθηκευτικού χώρου, στην εφαρμογή εκχωρείται το ποσό του ορίου που ζητήθηκε. Δεν χρειάζεται να ζητηθεί περισσότερη ποσόστωση/ quota μέχρι να υπάρξει πρόβλημα με το χώρο. Στην περίπτωση αυτή, ο καλύτερος τρόπος για ανάκτηση είναι να γίνει προσπάθεια για λειτουργία εγγραφής, να ληφθεί το QUOTA_EXCEEDED_ERR στην επανάκληση του σφάλματος και να ζητηθεί μόνιμος χώρος αποθήκευσης χρησιμοποιώντας την requestQuota.

3.6.8. FileEntry

Τα αρχεία στο σύστημα αρχείων sandbox αντιπροσωπεύονται από τη διεπαφή FileEntry. Ένα FileEntry περιέχει τους τύπους ιδιοτήτων και μεθόδων που θα περίμενε κανείς από ένα τυπικό σύστημα αρχείων.

Οι ιδιότητές του είναι οι εξής:

- **isFile:** Boolean. Σωστό αν η καταχώρηση είναι αρχείο.
- **isDirectory:** Boolean. Σωστό αν η καταχώρηση είναι κατάλογος.
- **name:** DOMString. Το όνομα της καταχώρισης, εξαιρουμένης της διαδρομής που οδηγεί σε αυτήν.
- **Fullpath:** DOMString. Η πλήρης απόλυτη διαδρομή από τη ρίζα στην είσοδο.
- **File system:** Σύστημα αρχείων. Το σύστημα αρχείων στο οποίο βρίσκεται η καταχώρηση.

Οι μέθοδοί του είναι οι ακόλουθοι:

- **getMetadata** (επιτυχής επιστροφή κλήσης, opt_errorCallback): Αναζητήστε μεταδεδομένα για αυτήν την καταχώρηση.

- **moveTo** (parentDirEntry, opt_newName, opt_successCallback, opt_errorCallback): Μετακίνηση μιας καταχώρησης σε διαφορετική θέση στο σύστημα αρχείων.
- **copyTo** (ParentDirEntry, opt_newName, opt_successCallback, opt_errorCallback): Αντιγράφει μια καταχώρηση σε διαφορετικό γονέα στο σύστημα αρχείων. Τα αντίγραφα του καταλόγου είναι πάντα αναδρομικά. Είναι σφάλμα να αντιγραφεί ένας κατάλογος μέσα στον εαυτό του ή σε κάποιον γονέα του εάν δεν δοθεί νέο όνομα.
- **toURL ();**Επιστρέφει ένα σύστημα αρχείων: η URL που μπορεί να χρησιμοποιηθεί για την αναγνώριση αυτού του αρχείου.
- **remove** (successCallback, opt_errorCallback): Διαγράφει ένα αρχείο ή έναν κατάλογο. Είναι σφάλμα η προσπάθεια διαγραφής του ριζικού καταλόγου ενός συστήματος αρχείων ή ενός καταλόγου που δεν είναι κενός.
- **getParent** (successCallback, opt_errorCallback): επιστρέφει το parent DirectoryEntry που περιέχει αυτήν την καταχώρηση. Εάν αυτή η καταχώρηση είναι ο ριζικός κατάλογος, ο κατάλογος-γονέας είναι ο ίδιος.
- **CreateWriter** (successCallback, opt_errorCallback)
- Δημιουργεί ένα νέο FileWriter που μπορεί να χρησιμοποιηθεί για την εγγραφή περιεχομένου σε αυτό το FileEntry.
- **file** (successCallback, opt_errorCallback): Επιστρέφει ένα αρχείο που αντιπροσωπεύει το FileEntry στην επιτυχή επανάκληση.

3.6.9. Δημιουργία αρχείου

Μετά το «Άνοιγμα συστήματος αρχείων», το Σύστημα Αρχείων που μεταβιβάζεται στην επιτυχή επανάκληση περιέχει τη ρίζα DirectoryEntry (ως fs.root). Για αναζήτηση ή δημιουργία αρχείου σε αυτόν τον κατάλογο, πρέπει να κληθεί η getFile(), περνώντας το όνομα του αρχείου που θα δημιουργηθεί. Για παράδειγμα, ο ακόλουθος κώδικας δημιουργεί ένα κενό αρχείο που ονομάζεται log.txt στον ριζικό κατάλογο.

Παράδειγμα. Δημιουργία αρχείου και εκτύπωση της τελευταίας τροποποίησης του χρόνου

function onFs(fs) {

```
fs.root.getFile(' log.txt', {create: true, exclusive: true},
function(fileEntry) {
// fileEntry.isFile === true
// fileEntry.name == 'log.txt'
// fileEntry.fullPath == '/log.txt'
fileEntry.getMetaData(function(md) {
console.log(md.modificationTime.toDateString());
}, onError);
},
onError
);
}
window.requestFileSystem(TEMPORARY, 1024*1024 /*1MB*/, onFs, onError);
```

Το πρώτο όρισμα για την getFile() μπορεί να είναι μια απόλυτη ή σχετική διαδρομή, αλλά πρέπει να είναι έγκυρη. Για παράδειγμα, είναι σφάλμα η δημιουργία αρχείου του οποίου ο άμεσος γονέας δεν υπάρχει. Το δεύτερο όρισμα είναι ένα αντικείμενο κυριολεκτικά που περιγράφει τη συμπεριφορά της συνάρτησης εάν το αρχείο δεν υπάρχει. Σε αυτό το παράδειγμα, το create: true δημιουργεί το αρχείο εάν δεν υπάρχει και εκπέμπει ένα σφάλμα εάν υπάρχει (αποκλειστικό: true). Διαφορετικά, εάν δημιουργηθεί: false, το αρχείο απλώς ανακτάται και επιστρέφεται. Από μόνη της, η αποκλειστική επιλογή δεν έχει κανένα αποτέλεσμα. Και στις δύο περιπτώσεις, τα περιεχόμενα του αρχείου δεν αντικαθίστανται. Απλώς λαμβάνεται μια καταχώρηση αναφοράς στο εν λόγω αρχείο.

3.6.10. Ανάγνωση Αρχείου

Η κλήση της getFile() ανακτά μόνο ένα FileEntry. Δεν επιστρέφει τα περιεχόμενα ενός αρχείου. Για αυτό, απαιτείται ένα αντικείμενο File και το FileReader API. Για να αποκτηθεί ένα Αρχείο, πρέπει να κληθεί η FileEn try.file(). Το πρώτο του όρισμα

είναι μια επιτυχής επανάκληση που μεταβιβάζεται στο αρχείο και το δεύτερο είναι μια επιστροφή κλήσης σφάλματος.

Προσθήκη δεδομένων σε αρχείο

Η προσθήκη δεδομένων σε ένα υπάρχον αρχείο είναι ασήμαντη με το FileWriter. Το πρόγραμμα εγγραφής μπορεί να επανατοποθετηθεί στο τέλος του αρχείου χρησιμοποιώντας seek(). Το Seek παίρνει μια μετατόπιση byte ως όρισμα, ορίζοντας τη θέση του συντάκτη του αρχείου σε αυτή τη μετατόπιση. Εάν η μετατόπιση είναι μεγαλύτερη από το μήκος του αρχείου, χρησιμοποιείται το τρέχον μήκος. Εάν η μετατόπιση είναι < 0, η θέση τίθεται πίσω από το τέλος του αρχείου.

Εισαγωγή Αρχείων

Για λόγους ασφαλείας, το API του συστήματος αρχείων HTML5 δεν επιτρέπει στις εφαρμογές να γράφουν δεδομένα εκτός του sandbox τους. Ως αποτέλεσμα αυτού του περιορισμού, οι εφαρμογές δεν μπορούν να μοιράζονται συστήματα αρχείων και δεν μπορούν να διαβάσουν ή να γράψουν αρχεία σε αυθαίρετους φακέλους στον σκληρό δίσκο του χρήστη, όπως ο φάκελος My Pictures ή My Music. Αυτό φέρνει τους προγραμματιστές σε μια δύσκολη θέση. Πώς εισάγει κανείς αρχεία σε μια εφαρμογή Ιστού εάν η εφαρμογή δεν μπορεί να έχει πρόσβαση στον πλήρη σκληρό δίσκο του χρήστη με όλα τα πολύτιμα αρχεία του;

Υπάρχουν τέσσερις τεχνικές για την εισαγωγή δεδομένων στο σύστημα αρχείων:

• Χρήση <input type="file">. Ο χρήστης επιλέγει αρχεία από μια τοποθεσία στον υπολογιστή του και η εφαρμογή αντιγράφει αυτά τα αρχεία στο σύστημα αρχείων HTML5 της εφαρμογής.

• Με μεταφορά και απόθεση HTML5. Ορισμένα προγράμματα περιήγησης υποστηρίζουν τη μεταφορά αρχείων από την επιφάνεια εργασίας στην καρτέλα του προγράμματος περιήγησης. Και πάλι, τα επιλεγμένα αρχεία θα αντιγραφούν στο σύστημα αρχείων HTML5.

Χρησιμοποιώντας <input type="file">

Ο πρώτος (και πιο συνηθισμένος) τρόπος εισαγωγής αρχείων σε μια εφαρμογή είναι η επανατοποθέτηση του <input type="file">. Στην περίπτωση αυτή όμως δεν ενδιαφέρει η μεταφόρτωση δεδομένων φόρμας—η τυπική χρήση μιας εισαγωγής

αρχείου. Αντίθετα, Μπορεί να χρησιμοποιηθεί το πρόγραμμα επιλογής εγγενών αρχείων του προγράμματος περιήγησης, να ζητηθεί από τους χρήστες να επιλέξουν αρχεία και να αποθηκευτούν αυτές τις επιλογές στην εφαρμογή.

Χρήση HTML5 Drag and Drop

Η δεύτερη μέθοδος για την εισαγωγή αρχείων είναι η Drag and Drop HTML5. Ένα πολύ ωραίο στοιχείο που δίνει το drag and drop είναι ένας οικείος τρόπος για τους χρήστες να εισάγουν δεδομένα στις εφαρμογές web.

Το Chrome, το Safari 5 και ο Firefox 4 επεκτείνουν τα συμβάντα Drag and Drop HTML5 επιτρέποντας τη μεταφορά αρχείων από την επιφάνεια εργασίας στο παράθυρο του προγράμματος περιήγησης. Στην πραγματικότητα, η διαδικασία για τη ρύθμιση των ακροατών συμβάντων για τη διαχείριση των απορριπτόμενων αρχείων είναι ακριβώς η ίδια με τη διαχείριση άλλων τύπων περιεχομένου. Η μόνη διαφορά είναι ο τρόπος πρόσβασης στα αρχεία στο πρόγραμμα χειρισμού πτώσης. Συνήθως, τα απορριπτόμενα δεδομένα διαβάζονται από την ιδιότητα dataTransfer του συμβάντος (ως dataTrans fer.getData()). Ωστόσο, κατά το χειρισμό αρχείων, τα δεδομένα διαβάζονται από το dataTransfer.files. Πρόκειται ουσιαστικά για το ισοδύναμο του Drag and Drop του προηγούμενου παραδείγματος χρησιμοποιώντας <input type="file">.

Χρησιμοποιώντας το XMLHttpRequest

Ένας τρίτος τρόπος εισαγωγής δεδομένων είναι η χρήση του XMLHttpRequest για τη λήψη απομακρυσμένων αρχείων. Η διαφορά μεταξύ αυτής της μεθόδου και των δύο πρώτων μεθόδων είναι ότι αυτή η επιλογή απαιτεί τα δεδομένα να υπάρχουν ήδη κάπου στο νέφος. Στις περισσότερες περιπτώσεις αυτό δεν είναι πρόβλημα. Ως προγραμματιστές ιστού, έχουν μάθει να ασχολούνται με απομακρυσμένα δεδομένα και να τα αντιμετωπίζουν συνεχώς.

Πολλές από τις βελτιώσεις που εισάγονται στο XMLHttpRequest Level 2 έχουν σχεδιαστεί για καλύτερη διαλειτουργικότητα με δυαδικά δεδομένα, blobs και αρχεία. Αυτά είναι πραγματικά καλά νέα για τους προγραμματιστές ιστού. Σημαίνει ότι μπορεί να μπει ένα τέλος σε τρελούς χειρισμούς συμβολοσειρών και επιρρεπείς σε σφάλματα παραβιάσεις κώδικα χαρακτήρων στις εφαρμογές.

3.6.11. Αφαίρεση αρχείων

Για να αφαιρεθεί ένα αρχείο από το σύστημα αρχείων, πρέπει να κληθεί η entry.remove(). Το πρώτο όρισμα αυτής της μεθόδου είναι μια συνάρτηση επανάκλησης μηδενικής παραμέτρου, η οποία καλείται όταν το αρχείο διαγραφεί με επιτυχία. Το δεύτερο είναι μια προαιρετική επανάκληση σφάλματος εάν παρουσιαστούν σφάλματα.

Παράδειγμα. Αφαίρεση αρχείου με όνομα

```
window.requestFileSystem(TEMPORARY, 1024*1024 /*1MB*/, function(fs) {
 fs.root.getFile('log.txt', {}, function(fileEntry) {
 fileEntry.remove(function() {
 console.log('File removed.');
}, onError);
 }, onError);
}, onError);
```

3.6.12. Το DirectoryEntry

Οι κατάλογοι στο σύστημα αρχείων sandbox αντιπροσωπεύονται από τη διεπαφή DirectoryEntry. Ένα DirectoryEntry περιέχει πολλές από τις ιδιότητες και τις μεθόδους που βρίσκονται στο FileEntry. Και οι δύο κληρονομούν από μια γενική διεπαφή εισόδου. Ωστόσο, περιλαμβάνει πρόσθετη μέθοδο για εργασία με αρχεία.

Οι ιδιότητες του είναι οι εξής:

- **isFile**: Boolean. Σωστό αν η καταχώρηση είναι αρχείο.
- **isDirectory**: Boolean. Σωστό αν η καταχώρηση είναι κατάλογος.
- **name**: DOMString. Το όνομα του καταλόγου, εξαιρουμένης της διαδρομής που οδηγεί σε αυτόν.
- **fullPath**: DOMString. Η πλήρης απόλυτη διαδρομή από τη ρίζα στον κατάλογο.
- **FileSystem**: FileSystem. Το σύστημα αρχείων στο οποίο βρίσκεται ο κατάλογος.

Οι μέθοδοί του είναι οι εξής:

getMetadata (successCallback, opt_errorCallback) : Αναζητά μεταδεδομένα για αυτόν τον κατάλογο.

moveTo (parentDirEntry, opt_newName, opt_successCallback, opt_errorCallback): Μετακινεί τον κατάλογο σε διαφορετική θέση στο σύστημα αρχείων.

copyTo (ParentDirEntry, opt_newName, opt_successCallback, opt_errorCallback): Αντιγράφει τον κατάλογο σε διαφορετικό γονέα στο σύστημα αρχείων. Τα αντίγραφα καταλόγου είναι πάντα αναδρομικά. Σφάλμα αποτελεί η αντιγραφή καταλόγου μέσα στον εαυτό του ή στον γονικό του εάν δεν παρέχεται νέο όνομα.

toURL (); : Επιστρέφει ένα σύστημα αρχείων URL που μπορεί να χρησιμοποιηθεί για την αναγνώριση αυτού του καταλόγου.

remove (successCallback, opt_errorCallback): Διαγράφει ένα αρχείο ή έναν κατάλογο. Είναι σφάλμα η προσπάθεια διαγραφής του ριζικού καταλόγου ενός συστήματος αρχείων ή ενός καταλόγου που δεν είναι κενός.

getParent (successCallback, opt_errorCallback): Επιστρέφει το γονικό DirectoryEntry που περιέχει αυτόν τον κατάλογο. Εάν αυτός ο κατάλογος είναι ο ριζικός κατάλογος, ο γονέας του είναι ο ίδιος.

createReader () : Δημιουργεί ένα νέο DirectoryReader για την ανάγνωση εγγραφών που σχετίζονται με αυτό το DirectoryEntry.

getFile (διαδρομή, επιλογέςObj, opt_successCallback, opt_errorCallback) : Δημιουργεί ή αναζητά ένα FileEntry. Το πρώτο όρισμα είναι μια διαδρομή που αντιπροσωπεύει απόλυτη ή σχετική διαδρομή από αυτόν τον κατάλογο. Το δεύτερο όρισμα είναι ένα αντικείμενο που περιγράφει τη συμπεριφορά αυτής της μεθόδου εάν το αρχείο δεν υπάρχει. Αν create: truth και exlusive: true εμφανίζεται σφάλμα εάν το αρχείο υπάρχει ήδη στη διαδρομή. Εάν create: truth και exlusive: false, το αρχείο θα δημιουργηθεί. Αν υπάρχει ήδη, δεν θα πεταχτεί κανένα λάθος. Τέλος, εάν create: false το αρχείο επιστρέφεται εάν υπάρχει και εμφανίζεται ένα σφάλμα εάν δεν υπάρχει. Κατά τη λήψη ενός αρχείου, η αποκλειστική σημαία αγνοείται. Εάν είναι επιτυχής, επιστρέφεται ένα FileEntry στην επανάκληση.

getDirectory (διαδρομή, επιλογέςObj, opt_successCallback, opt_errorCallback) : Δημιουργεί ή αναζητά ένα DirectoryEntry. Η σημασιολογία αυτής της μεθόδου είναι

ίδια με την getFile(), με τη διαφορά ότι μια DirectoryEntry μεταβιβάζεται στην επιτυχή επανάκληση.

removeRecursively (successCallback, opt_errorCallback) : Διαγράφει αναδρομικά αυτόν τον κατάλογο και όλα τα περιεχόμενά του. Παρουσιάζεται σφάλμα εάν κάποιος προσπαθήσει να καταργήσει τον ριζικό κατάλογο ενός συστήματος αρχείων. Εάν παρουσιαστεί σφάλμα ενώ αυτή η μέθοδος βρίσκεται σε εξέλιξη, ορισμένα από τα περιεχόμενα του καταλόγου ενδέχεται να μην διαγραφούν.

3.6.13. Δημιουργία καταλόγων

Ένα API συστήματος αρχείων πρέπει να υποστηρίζει ιεραρχίες προσαρμοσμένων φακέλων. Μπορεί κανείς να δημιουργήσει ή να ανακτήσει έναν κατάλογο χρησιμοποιώντας το DirectoryEntry.getDirectory(). Η σημασιολογία του είναι ακριβώς η ίδια με την FileEntry.getFile(). Η πρώτη παράμετρος είναι μια συμβολοσειρά που αντιπροσωπεύει μια απόλυτη ή σχετική διαδρομή του καταλόγου προς αλληλεπίδραση. Για παράδειγμα, ο ακόλουθος κώδικας δημιουργεί έναν κατάλογο με το όνομα MyPictures στον ριζικό κατάλογο.

Παράδειγμα. Δημιουργία φακέλου

```
function onFs(fs) {
 fs.root.getDirectory(' MyPictures', {create: true}, function(dirEntry) {
 // dirEntry.isFile === false
 // dirEntry.isDirectory === true
 // dirEntry.name == 'MyPictures'
 // dirEntry.fullPath == '/MyPictures'
 }, onError);
}
window.requestFileSystem(TEMPORARY, 1024*1024 /*1MB*/, onFs, onError);
```

Το API εκθέτει μια διεπαφή DirectoryReader για την ανάγνωση των εγγραφών σε έναν κατάλογο.

Μέθοδοι

readEntries (successCallback, opt_errorCallback)

Επιστρέφει μια λίστα με καταχωρήσεις σε αυτόν τον κατάλογο. Αυτή η μέθοδος πρέπει να κληθεί μέχρι να επιστραφεί ένας κενός πίνακας.

Για να παρατεθούν όλα τα αρχεία και οι φάκελοι σε ένα DirectoryEntry, πρέπει πρώτα να δημιουργηθεί ένα DirectoryReader (μια σύγχρονη λειτουργία):

var reader = directoryEntry.createReader();

Στη συνέχεια, πρέπει να κληθεί η readEntries(). Το API δεν εγγυάται ότι όλες οι εγγραφές επιστρέφονται σε μία μόνο κλήση στο readEntries(). Πρέπει να συνεχίσει να καλείται η readEntries() μέχρι να μην επιστρέφονται άλλα αποτελέσματα. Το παρακάτω παράδειγμα αποδεικνύει ακριβώς αυτό.

Παράδειγμα. Καταχώρηση των εγγραφών σε έναν κατάλογο

```
<!DOCTYPE html>

<html>

<head>

 <meta charset="utf-8">

 <title>Listing the entries in a directory</title>

</head>

<body>

<ul id="filelist"></ul>

<script>

// Take care of vendor prefixes.

window.requestFileSystem = window.requestFileSystem ||

window.webkiRequestFileSystem;

/**

 * Returns an Array from a NodeList or other array-like object.

 *
```

```
 * @param {NodeList} list An array-like object to transform into an array.
 * @return {Array} The list as an array.
 */
function toArray(list) {
 return Array.prototype.slice.call(list || [], 0);
}
/**
 * Renders a list of file/folder entries.
 *
 * @param {Array<FileEntry|DirectoryEntry>} entries A list of file/folders.
 */
function listResults(entries) {
 // Use a document fragment. Will cause only one reflow on append :)
 var fragment = document.createDocumentFragment();
 entries.forEach(function(entry, i) {
 var img = entry.isDirectory ? '<img src="folder-icon.gif">' :
 '<img src="file-icon.gif">';
 var li = document.createElement('li');
 li.innerHTML = [img, '<span>', entry.name, '</span>'].join('');
 fragment.appendChild(li);
 });
 document.querySelector('#filelist').appendChild(fragment);
}
window.requestFileSystem(TEMPORARY, 1024*1024 /*1MB*/, function(fs) {
```

```
var dirReader = fs.root.createReader();
var entries = [];
// Call the reader.readEntries() until no more results are returned.
var readEntries = function() {
dirReader.readEntries(function(results) {
// If no more results are returned, we're done.
if (!results.length) {
// Sort list by name of entry.
entries.sort(function(a, b) {
return a.name < b.name ? -1 :
b.name < a.name ? 1 : 0;
});
listResults(entries); // Render the list.
} else {
// Add in these results to the current list.
entries = entries.concat(toArray(results));
readEntries();
}
}, onError);
};
readEntries(); // Start reading the directory.
}, onError);
</script>
</body>
```

</html>

Υπάρχουν δύο τρόποι για να αφαιρεθεί ένα DirectoryEntry από το σύστημα αρχείων, remove() και removeRecursively().

Η πρώτη μέθοδος έχει την ίδια σημασιολογία με την FileEntry.remove(), λαμβάνοντας μια προαιρετική επανάκληση μετά την επιτυχή διαγραφή. Ωστόσο, εάν επιχειρήσει κάποιος να διαγράψει έναν κατάλογο που δεν είναι κενός, το API εμφανίζει ένα σφάλμα. Κάνοντας την αναλογία με το UNIX, παρουσιάζεται το ίδιο σφάλμα κατά την κλήση του rmdir σε έναν μη κενό φάκελο:

```
dirEntry.remove(function() {
 console.log('Directory removed.');
}, onError);
```

4. Πρακτικό Μέρος

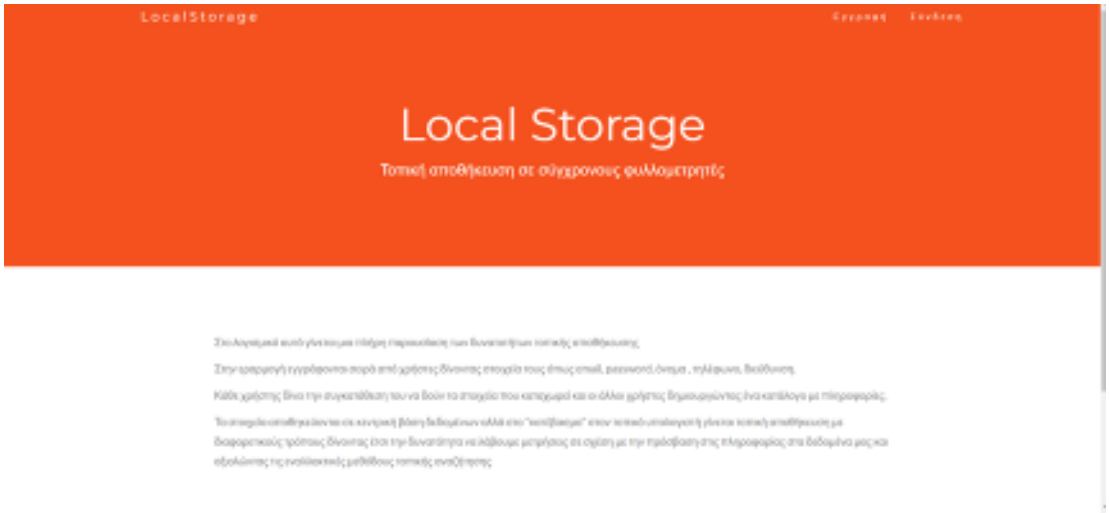
4.1. Περιγραφή Εφαρμογής

Η εφαρμογή περιλαμβάνει τα παρακάτω:

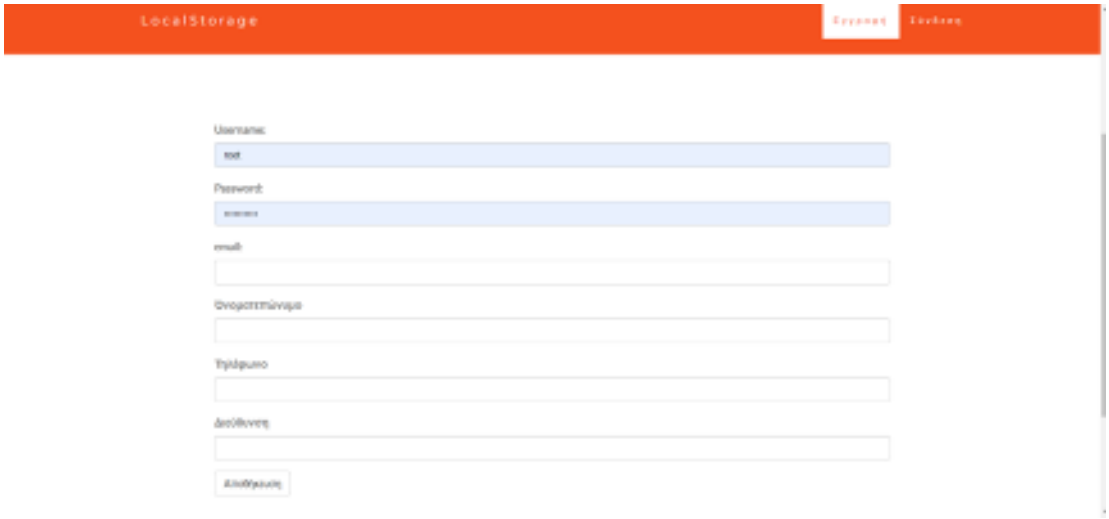
1. Κεντρική βάση δεδομένων που αποθηκεύονται τα δεδομένα για να μπορούν να ανακτηθούν τοπικά και να δοκιμαστούν με διαφορετικές μεθόδους τοπικής αποθήκευσης.
2. Σύνδεση της βάσης δεδομένων με API στοιχεία
3. Περιβάλλον διαχείρισης σε HTML/CSS Javascript για το Front End

Στην εφαρμογή γίνονται τα παρακάτω:

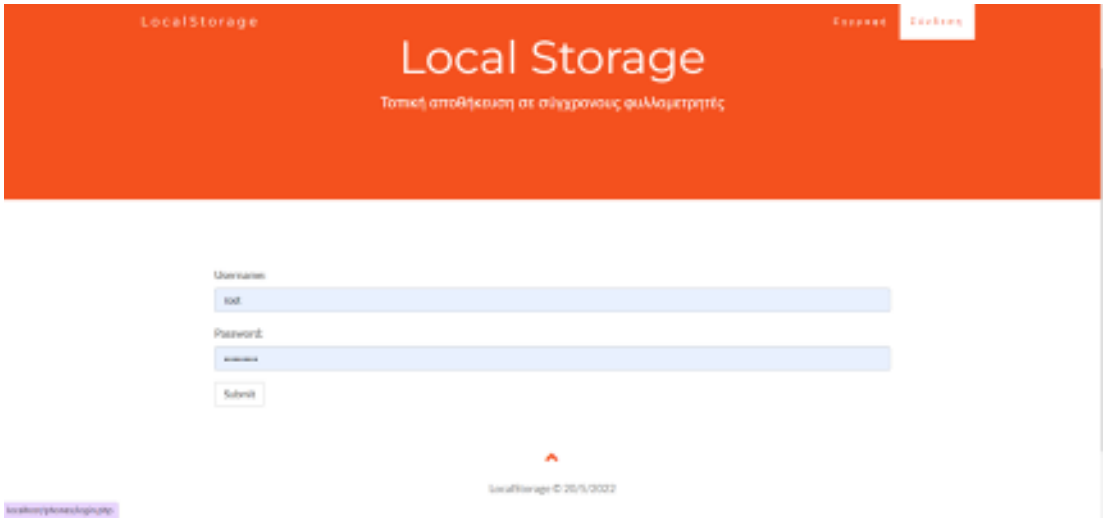
1. Αρχικά στο χρήστη εμφανίζεται η παρακάτω εικόνα:



Ο χρήστης μπορεί να κάνει εγγραφή και να προστεθεί στη λίστα των χρηστών της εφαρμογής.



Μετά την εγγραφή του μπορεί να κάνει Login



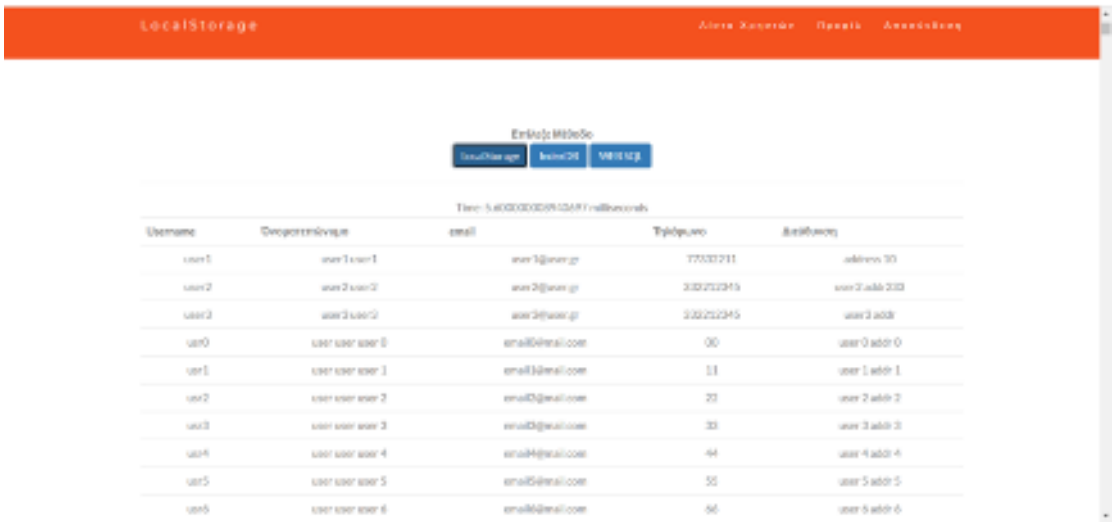
Όταν συνδεθεί παίρνει σαν αποτέλεσμα την παρακάτω εικόνα:

Αν θέλει να αλλάξει τα στοιχεία τότε πατάει στο Προφίλ και παίρνει την παρακάτω οθόνη όπου εκεί μπορεί να κάνει αλλαγές στα στοιχεία του προφίλ του.

Τα παραπάνω γίνονται όλα στον server δηλαδή έχουμε server side data store όπου τα στοιχεία μας αποθηκεύονται με χρήση PHP και MYSQL στην βάση με το όνομα dbphones που περιέχει τον πίνακα phone που έχει σαν πεδία:

id, onoma, email, phone, addr, password.

Ο χρήστης μπορεί να επιλέξει λίστα χρηστών όπου εκεί μπορεί να πάρει τα στοιχεία των εγγεγραμμένων χρηστών επιλέγοντας μια από τις 3 διαφορετικές μεθόδους Client Data Store. Σε κάθε περίπτωση μετράμε τον χρόνο που γίνεται η ανάκτηση των στοιχείων ώστε να συγκρίνουμε τις μεθόδους μεταξύ τους.



Το παραπάνω αποτέλεσμα είναι για 1000 χρήστες που ανακτώνται με την μέθοδο LocalStorage.

4.2. Cliend Data Store

Οι μέθοδοι που υποστηρίζονται είναι :

4.2.1. LocalStorage

Το Local Storage ή Web Storage παρέχει δύο παρόμοια API για τον ορισμό ζευγών ονόματος/τιμής.

- window.localStorage για αποθήκευση μόνιμων δεδομένων και
- window.sessionStorage για διατήρηση δεδομένων μόνο για περιόδους σύνδεσης όσο η καρτέλα του προγράμματος περιήγησης παραμένει ανοιχτή

Το βασικό πλεονέκτημα της μεθόδου είναι ότι έχουμε ένα απλό API ζεύγους ονόματος/τιμής, επιλογές συνεδρίας και μόνιμης αποθήκευσης, καλή υποστήριξη των προγραμμάτων περιήγησης.

Τα βασικά μειονεκτήματα είναι ότι υποστηρίζει μόνο συμβολοσειρές και έτσι απαιτεί parsing για τα στοιχεία JSON και δεν υποστηρίζει μη δομημένα δεδομένα χωρίς συναλλαγές, ευρετηρίαση ή αναζήτηση.

Το Web Storage είναι ιδανικό για απλούστερες, μικρότερες τιμές. Είναι λιγότερο πρακτικό για την αποθήκευση μεγάλου όγκου δομημένων πληροφοριών.

Στην περίπτωση μας χρησιμοποιήσαμε την μέθοδο μέσα από το συγκεκριμένο script που δίνεται στην σελίδα list.php

```
$.get("phpapi/api1.php?query=3", function(res)
                    {



                                localStorage.setItem('data', res);
                                var startTime = performance.now();

                                var data= JSON.parse(localStorage.getItem('data'));


                                s="<table class=table>";


s+="<tr><th>Username</th><th>Όνοματεπώνυμο</th><th>email</th><th>Τηλέφωνο</
th><th>Διεύθυνση</th></tr>";
                                for (i=0;i<data.length;i++)
                                {

s+="<tr><td>"+data[i].username+"</td><td>"+data[i].onoma+"</td><td>"+data[i].email+
"</td><td>"+data[i].phone+"</td><td>"+data[i].addr+"</td></tr>";
                                }
                                s+="</table>";
                                $("#data").html(s);

                                var endTime = performance.now();
                                $("#tmexec").html((endTime-startTime)+" milliseconds");
                    });
```

Ουσιαστικά παίρνουμε τα δεδομένα από την βάση τα διαβάζουμε από το API και στην συνέχεια αποθηκεύουμε.

Μετά τα παίρνουμε από την μεταβλητή που τα αποθηκεύσαμε , τα μετατρέπουμε σε JSON Object Array και τα εμφανίζουμε.

Σε υπολογιστή με χαρακτηριστικά i7, 16Gb Ram , Δίσκο SSD έχουμε χρόνο ανάκτησης στα 5ms σε 1000 αποθηκευμένα ονόματα στην βάση.

4.2.2. IndexedDB

Το IndexedDB προσφέρει ένα API χαμηλού επιπέδου που μοιάζει με NoSQL για την αποθήκευση μεγάλων όγκων δεδομένων. Τα στοιχεία μπορούν να χρησιμοποιήσουν ευρετήριο, να ενημερωθούν χρησιμοποιώντας συναλλαγές και να αναζητηθούν χρησιμοποιώντας ασύγχρονες μεθόδους.

Το IndexedDB API είναι πολύπλοκο και απαιτεί ιδιαίτερους τρόπους αντιμετώπισης.

Είναι πιο κατάλληλη όταν υπάρχουν πολλά ερωτήματα στα δεδομένα και απαιτούν συνεχείς αλλαγές.

Το αντίστοιχο κομμάτι javascript που υλοποιήσαμε την διαδικασία ανάκτησης των δεδομένων , την εγγραφή και προβολή τους είναι το παρακάτω:

```
$.get("phpapi/api1.php?query=3", function(res)
                {




                                var data=JSON.parse(res);

                                var indexedDB = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.msIndexedDB || window.shimIndexedDB;

                                // Open (or create) the database
                                var open = indexedDB.open("MyDatabase", 1);

                                open.onupgradeneeded = function() {
                                        var db = open.result;
                                        var store = db.createObjectStore("MyObjectStore", {keyPath: "id"});

                                };

                                open.onsuccess = function() {
                                                var db = open.result;
                                                var tx = db.transaction("MyObjectStore", "readwrite");
                                                var store = tx.objectStore("MyObjectStore");

                                                for (var i in data)
```

```
                                        {
                                                store.put(data[i]);
                                        }

                                        var getAll = store.getAll();
                                        var startTime = performance.now();
                                         getAll.onsuccess = function() {
                                                        dt=getAll.result;

                                                        s="<table class=table>";


s+="<tr><th>Username</th><th>Όνοματεπώνυμο</th><th>email</th><th>Τηλέφωνο</
th><th>Διεύθυνση</th></tr>";
                                                        for (i=0;i<dt.length;i++)
                                                        {
                                                                d=dt[i];

s+="<tr><td>"+d.username+"</td><td>"+d.onoma+"</td><td>"+d.email+"</
td><td>"+d.phone+"</td><td>"+d.addr+"</td></tr>";
                                                        }
                                                        s+="</table>";
                                                        $("#data").html(s);
                                                        var endTime =
performance.now();
                                                $("#tmexec").html((endTime-
startTime)+" milliseconds");

                                                };
                                         tx.oncomplete = function() {
                                                db.close();
                                        };
                                }




        });
```

Σε υπολογιστή με χαρακτηριστικά i7, 16Gb Ram , Δίσκο SSD έχουμε χρόνο ανάκτησης στα 41ms σε 1000 αποθηκευμένα ονόματα στην βάση.

Αν και ο χρόνος φαίνεται αρκετά μεγαλύτερος , όταν θα χρειαστεί για πολύ μεγάλου όγκου δεδομένα η πρώτη ανάκτηση μπορεί να είναι αργή αλλά στην συνέχεια η πρόσβαση σε αυτά γίνεται ιδιαίτερα γρήγορα.

4.2.3. WEB SQL

Το WebSQL είναι μια προσπάθεια να φέρει στο πρόγραμμα περιήγησης την δυνατότητα χειρισμού των δεδομένων σαν βάσης δεδομένων τύπου SQL.

Ο Chrome και ορισμένες εκδόσεις του Safari υποστηρίζουν την τεχνολογία, αλλά ο Mozilla και τη Microsoft χρησιμοποιούν αποκλειστικά το IndexedDB.

Τα βασικά Πλεονεκτήματα του είναι:

- Είναι σχεδιασμένο για ισχυρή αποθήκευση και πρόσβαση δεδομένων από την πλευρά του πελάτη.
- Είναι γνώριμη η σύνταξη SQL που χρησιμοποιείται συχνά από προγραμματιστές από την πλευρά του διακομιστή.

Τα βασικά Μειονεκτήματα είναι:

- Είναι περιορισμένη και προβληματική υποστήριξη στους Φυλλομετρητές.
- Υπάρχει σχετικά ασυνεπής σύνταξη SQL σε όλα τα προγράμματα περιήγησης.
- Αν και λειτουργεί ασύγχρονα υπάρχει σχετικά κακή απόδοση στην επανάκληση δεδομένων.

Η υλοποίηση με WEB SQL στην περίπτωση μας έγινε όπως παρακάτω:

```
$.get("phpapi/api1.php?query=3", function(res)
{
	const db2 = openDatabase('dbphones', '1.0', 'phones db', 1024 *
1024);
	var data=JSON.parse(res);

	db2.transaction( t => {
		t.executeSql("CREATE TABLE IF NOT EXISTS
phones(username,password,onoma,email,phone,addr)" );
		t.executeSql("delete from phones" );

	});

	let sql="insert into phones
(username,password,onoma,email,phone,addr) values ";
		comma="";
	for (i=0;i<data.length;i++)
	{

		sql+=comma+"('"+data[i].username+"',
'"+data[i].password+"','"+data[i].onoma+"','"+data[i].email+"','"+data[i].phone+"','"+data[i
```

```
].addr+"')";
                                        comma=",";

                                }
                                db2.transaction( t => {

                                        t.executeSql(sql );
                                                console.log(sql);
                                        });


                                var startTime = performance.now();
                                db2.transaction( t => {

                                        t.executeSql(
                                                "SELECT * FROM phones",
                                                [],
                                                (t, results) => {


                                                        var len = results.rows.length, i;


                                                                s="<table class=table>";


s+="<tr><th>Username</th><th>Όνοματεπώνυμο</th><th>email</th><th>Τηλέφωνο</
th><th>Διεύθυνση</th></tr>";
                                                                for (i=0;i<len;i++)
                                                                {
                                                                        d=results.rows.item(i);

s+="<tr><td>"+d.username+"</td><td>"+d.onoma+"</td><td>"+d.email+"</
td><td>"+d.phone+"</td><td>"+d.addr+"</td></tr>";
                                                                }
                                                                s+="</table>";



                                                                $("#data").html(s);
                                                                var endTime =
performance.now();
                                                                $("#tmexec").html((endTime-
startTime)+" milliseconds");
                                                }
                                         );

                                });
```

```
                });
        }
```

Σε υπολογιστή με χαρακτηριστικά i7, 16Gb Ram , Δίσκο SSD έχουμε χρόνο ανάκτησης στα 38ms σε 1000 αποθηκευμένα ονόματα στην βάση.

Συμπεράσματα

Η αποθήκευση από την πλευρά του πελάτη λειτουργεί με παρόμοιες αρχές, αλλά έχει διαφορετικές χρήσεις. Αποτελείται από JavaScript API που σας επιτρέπουν την αποθήκευση τοπικά δηλαδή δεδομένα στον client(στον υπολογιστή του χρήστη) και στη συνέχεια να γίνεται ανάκτηση όποτε τα χρειαζόμαστε. Αυτό έχει πολλές διαφορετικές χρήσεις, όπως την εξατομίκευση προτιμήσεων ιστότοπου (π.χ. εμφάνιση της επιλογής ενός χρήστη για προσαρμοσμένα γραφικά στοιχεία, συνδυασμό χρωμάτων ή μέγεθος γραμματοσειράς), τη συνεχή προηγούμενη δραστηριότητα ιστότοπου (π.χ. αποθήκευση των περιεχομένων ενός καλαθιού αγορών από μια προηγούμενη περίοδο σύνδεσης, ανάμνηση εάν ένας χρήστης ήταν προηγουμένως συνδεδεμένος), την αποθήκευση δεδομένων και στοιχείων τοπικά, ώστε ένας ιστότοπος να είναι πιο γρήγορος (και ενδεχομένως λιγότερο δαπανηρός) στη λήψη ή να μπορεί να χρησιμοποιηθεί χωρίς σύνδεση δικτύου καθώς και την αποθήκευση εγγράφων που δημιουργούνται από εφαρμογές Ιστού τοπικά για χρήση εκτός σύνδεσης

Συχνά ο χώρος αποθήκευσης από την πλευρά του πελάτη και τον διακομιστή χρησιμοποιούνται μαζί. Για παράδειγμα, θα μπορούσατε να γίνεται λήψη μιας δέσμης αρχείων μουσικής (ίσως χρησιμοποιούνται από ένα παιχνίδι web ή μια εφαρμογή αναπαραγωγής μουσικής), να αποθηκεύονται σε μια βάση δεδομένων από την πλευρά του πελάτη και να παίζουν όπως απαιτείται. Ο χρήστης θα πρέπει να κατεβάσει τα αρχεία μουσικής μόνο μία φορά — και μόνο σε αλλαγές σε επόμενες επισκέψεις θα ανακτηθούν από τη βάση δεδομένων. Γνωστές μεθόδοι είναι η χρήση βάσης δεδομένων, αποθήκευση σε τοπικά αρχεία, χρήση τοπικών session κ.α.

Στην παρούσα εργασία δημιουργήσαμε μια εφαρμογή μέσω ενός κεντρικού site σε ένα server (τοπικό server Apache και Mysql – XAMPP) που υποστηρίζεται από μια βάση δεδομένων όπου κάθε χρήστης έχει τη δυνατότητα να συνδέεται και να καταχωρεί εγγραφές, όπως διάφορα στοιχεία επαφών όπως όνομα, επώνυμο, τηλέφωνο , διεύθυνση. Κάθε χρήστης κάνει login και μπορεί να κατεβάσει τα

στοιχεία των επαφών που έχει αποθηκεύσει. Θεωρώντας ότι τα στοιχεία είναι πάρα πολλά, αρχικά πραγματοποιήθηκε μια προσωρινή τοπική αποθήκευση ώστε να έχει τη δυνατότητα ο χρήστης να τα χειρίζεται τοπικά (αναζητήσεις κ.λ.π) . Νέες εγγραφές και ανανεώσεις γίνονται ταυτόχρονα και στην τοπική βάση και στην βάση στον server.

Οι φυλλομετρητές που χρησιμοποιήθηκαν είναι οι πιο δημοφιλής (Microsoft Edge , Chrome, Mozilla), ενώ οι μετρήσεις που έγιναν αφορούσαν τη μέση απόκριση σε χρόνο σε αναζητήσεις επαφών, το χρόνο φόρτωσης όλης της πληροφορίας ανά χρήστη, το μέσο χρόνο ανά εγγραφή ανά φόρτωση (όταν θέλουμε να φορτώσουμε όλο το αρχείο του χρήστη) , τη χρήση τοπικής μνήμης. Οι περιπτώσεις που αναλύθηκαν ήταν σε ένα τοπικό αρχείο, στην τοπική βάση Non-SQL με την χρήση της IndexedDB και με χρήση του Cache API. Ουσιαστικά, στην τοπική σελίδα συνυπάρχουν 3 εκδόσεις του site που εκτελέστηκαν και πήραμε μετρήσεις από τις 3 αυτές περιπτώσεις. Στο τέλος έγινε κριτική αξιολόγηση με βάση τα αποτελέσματα από τις μετρήσεις για τις μεθόδους που εφαρμόστηκαν.

Το Local Storage ή Web Storage παρέχει τα API για τον ορισμό ζευγών ονόματος/τιμής window.localStorage (για αποθήκευση μόνιμων δεδομένων) και window.sessionStorage (για διατήρηση δεδομένων μόνο για περιόδους σύνδεσης). Το πλεονέκτημα της μεθόδου είναι ότι έχουμε ένα απλό API ζεύγους ονόματος/τιμής, επιλογές συνεδρίας και μόνιμης αποθήκευσης, καλή υποστήριξη των προγραμμάτων περιήγησης, ενώ το μειονεκτήματα είναι ότι υποστηρίζει μόνο συμβολοσειρές και έτσι απαιτεί parsing για τα στοιχεία JSON και δεν υποστηρίζει μη δομημένα δεδομένα χωρίς συναλλαγές, ευρετηρίαση ή αναζήτηση. Το Web Storage είναι ιδανικό για απλούστερες, μικρότερες τιμές. Είναι λιγότερο πρακτικό για την αποθήκευση μεγάλου όγκου δομημένων πληροφοριών. Από τις μετρήσεις που λάβαμε, σε υπολογιστή με χαρακτηριστικά i7, 16Gb Ram , Δίσκο SSD έχουμε χρόνο ανάκτησης στα 5ms σε 1000 αποθηκευμένα ονόματα στην βάση.

Το IndexedDB προσφέρει ένα API χαμηλού επιπέδου που μοιάζει με NoSQL για την αποθήκευση μεγάλων όγκων δεδομένων. Από τις μετρήσεις που πραγματοποιήθηκαν σε υπολογιστή με τα ίδια χαρακτηριστικά με αυτόν της μέτρησης που αφορούσε την περίπτωση του Web Storage, έχουμε χρόνο ανάκτησης στα 41ms σε 1000 αποθηκευμένα ονόματα στην βάση. Αν και ο χρόνος φαίνεται αρκετά μεγαλύτερος ,

όταν θα χρειαστεί για πολύ μεγάλου όγκου δεδομένα η πρώτη ανάκτηση μπορεί να είναι αργή αλλά στην συνέχεια η πρόσβαση σε αυτά γίνεται ιδιαίτερα γρήγορα.

Το WebSQL είναι μια προσπάθεια να φέρει στο πρόγραμμα περιήγησης την δυνατότητα χειρισμού των δεδομένων σαν βάσης δεδομένων τύπου SQL. Ο Chrome και ορισμένες εκδόσεις του Safari υποστηρίζουν την τεχνολογία, αλλά ο Mozilla και τη Microsoft χρησιμοποιούν αποκλειστικά το IndexedDB. Από της μετρήσεις στον ίδιο υπολογιστή και των προηγούμενων περιπτώσεων, είχαμε χρόνο ανάκτησης στα 38ms σε 1000 αποθηκευμένα ονόματα στην βάση.

Βιβλιογραφία

Benson, E., Marcus, A., Karger, D., & Madden, S. (2010, April). Sync kit: a persistent client-side database caching toolkit for data intensive websites. In *Proceedings of the 19th international conference on World wide web* (pp. 121-130).

Cannon, B., & Wohlstadter, E. (2010, April). Automated object persistence for JavaScript. In *Proceedings of the 19th international conference on World wide web* (pp. 191-200).

Cha, S. H., & Yun, Y. (2013). Smartphone Application Development using HTML5-based Cross-Platform Framework.

Connolly, T. M., & Begg, C. E. (2005). *Database systems: a practical approach to design, implementation, and management*. Pearson Education.

Dean, J., & Ghemawat, S. (2011). leveldb–A fast and lightweight key/value database library by Google.

Harrington, J. L. (2002). *Relational database design clearly explained*. Elsevier.

Hickson, I. (2010). HTML5 (including next generation additions still in development). *Draft Standard. Web Hypertext Application Technology Working Group (WHATWG)*, 150.

Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, D. E., O'Connor, E., & Pfeiffer, S. HTML5; Recommendation, W3C, 28.10. 2014.

Ijtihadie, R. M., Chisaki, Y., Usagawa, T., Cahyo, H. B., & Affandi, A. (2010, November). Offline web application and quiz synchronization for e-learning activity for mobile browser. In *TENCON 2010-2010 IEEE Region 10 Conference* (pp. 2402-2405). IEEE.

Kessin, Z. (2011). *Programming HTML5 applications: building powerful cross-platform environments in JavaScript*. " O'Reilly Media, Inc.".

Kimak, S., & Ellman, J. (2013). Performance testing and comparison of client side databases versus server side. *Northumbria University*.

Kimak, S., & Ellman, J. (2015, December). The role of HTML5 IndexedDB, the past, present and future. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)* (pp. 379-383). IEEE.

Laine, M. (2012). Client-side storage in web applications. *Aalto University*.

Leavitt, N. (2010). Will NoSQL databases live up to their promise?. *Computer*, *43*(2), 12-14.

Leblon, R. (2010). Building advanced, offline web applications with HTML 5. *UPC-Barcelona Tech*.

Lonergan, I. P. (2012). *Anyansi, Onyedikachi Jeffrey* (Doctoral dissertation, WORCESTER POLYTECHNIC INSTITUTE).

Mehta, N., Sicking, J., Graff, E., Popescu, A., & Orlow, J. (2012). Indexed Database API. W3C Working Draft. *World Wide Web Consortium*.

Mehta, N., Sicking, J., Graff, E., Popescu, A., Orlow, J., & Bell, J. (2015). Indexed Database API: W3C Recommendation 08 January 2015.

Mickens, J. (2010). Silo: Exploiting {JavaScript} and {DOM} Storage for Faster Page Loads. In *USENIX Conference on Web Application Development (WebApps 10)*.

Shashank, T. (2011). Professional NoSQL.

Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Communications of the ACM*, *53*(4), 10-11. management system. *Lainattu*, *5*, 2014.

Strozzi, C. (1998). NoSQL: A relational database

Thalheim, B. (2013). *Entity-relationship modeling: foundations of database technology*. Springer Science & Business Media.

Tudorica, B. G., & Bucur, C. (2011, June). A comparison between several NoSQL databases with comments and notes. In *2011 RoEduNet international conference 10th edition: Networking in education and research* (pp. 1-5). IEEE.

West, W., & Pulimood, S. M. (2012). Analysis of privacy and security in HTML5 web storage. *Journal of Computing Sciences in Colleges*, *27*(3), 80-87.