ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ
ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΜΜΕ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΙΣΤΟΤΟΠΟΥ ΚΟΙΝΩΝΙΚΗΣ ΔΙΚΤΥΩΣΗΣ

BARDHI JORGO (ΜΠΑΡΔΙ ΓΙΩΡΓΟ)

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ: ΔΡ. ΑΘΑΝΑΣΙΟΣ ΚΟΥΤΡΑΣ

ΠΥΡΓΟΣ, 2018

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η πτυχιακή εργασία με θέμα:

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΙΣΤΟΤΟΠΟΥ ΚΟΙΝΩΝΙΚΗΣ ΔΙΚΤΥΩΣΗΣ

του φοιτητή του Τμήματος ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΜΜΕ
ΜΠΑΡΔΙ ΓΙΩΡΓΟ
Α.Μ: 1745

παρουσιάστηκε δημόσια και εξετάσθηκε στο Τμήμα ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΜΜΕ στις

_____ / _____ / _____

Ο ΕΠΙΒΛΕΠΩΝ

Δρ. ΑΘΑΝΑΣΙΟΣ ΚΟΥΤΡΑΣ

Ο ΠΡΟΕΔΡΟΣ ΤΟΥ ΤΜΗΜΑΤΟΣ

ΚΟΥΓΙΑΣ ΙΩΑΝΝΗΣ

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΜΗ ΛΟΓΟΚΛΟΠΗΣ

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Ακόμα δηλώνω ότι αυτή η γραπτή εργασία προετοιμάστηκε από εμένα προσωπικά και αποκλειστικά και ειδικά για την συγκεκριμένη πτυχιακή εργασία και ότι θα αναλάβω πλήρως τις συνέπειες εάν η εργασία αυτή αποδειχθεί ότι δεν μου ανήκει.

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΣΠΟΥΔΑΣΤΗ 1	**ΑΜ**	**ΥΠΟΓΡΑΦΗ**
ΜΠΑΡΔΙ ΓΙΩΡΓΟ	1745	

ΕΥΧΑΡΙΣΤΙΕΣ

Αισθάνομαι την ανάγκη να ευχαριστήσω την οικογένεια μου
για όλα όσα μου έχουν προσφέρει στη διάρκεια των μαθητικών και
φοιτητικών μου χρόνων και την υποστήριξη τους σε κάθε μου βήμα.

Ιδιαιτέρως θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου
κ. Αθανάσιο Κούτρα για την εμπιστοσύνη και την αφιέρωση πολύτιμου χρόνου ώστε να ολοκληρωθεί η εργασία αυτή.

ΠΡΟΛΟΓΟΣ

Ο ταχύτατα αναπτυσσόμενος κόσμος της πληροφορικής έφερε την επανάσταση στον τρόπο που επικοινωνούν οι άνθρωποι μεταξύ τους. Στα προηγούμενα χρόνια οι άνθρωποι ως μόνο τρόπο επικοινωνίας είχαν το τηλέφωνο πλέων με την ραγδαία τεχνολογική έξαρση μπορούν να επικοινωνούν και να μοιράζονται τις στιγμές τους μέσω των κοινωνικών δικτύων.

Αυτό προσπαθεί να παρουσιάσει και το παρών σύγγραμμα ένα νέο κοινωνικό δίκτυο για να μοιράζονται οι χρήστες με τους φίλους τους φωτογραφίες και να αποτυπώνουν τις καθημερινές τους στιγμές.

Ο αναγνώστης του παρόντος συγγράμματος θα είναι σε θέση να κατανοήσει πλήρως πως δομείται από το μηδέν ένα πολύπλοκο κοινωνικό δίκτυο, θα κατανοήσει το πως λειτουργεί ο αντικειμενοστραφής προγραμματισμός της γλώσσας προγραμματισμού Ruby αλλά και την γλώσσας JavaScript και θα είναι σε θέση να κατανοήσει πως δομούνται αυτή την στιγμή όλες οι σύγχρονες ιστοσελίδες μέσω της HMTL και CSS αλλά και το πώς τροποποιούνται για να χωρέσουν σε μια μικρή οθόνη κινητού δηλαδή το τι είναι το responsive design και το πως υλοποιείται στην πράξη.

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια βλέπουμε όλο και περισσότερα social networks να εμφανίζονται και οι χρήστες να τα χρησιμοποιούν όλο και περισσότερο για αυτό και εμείς προσπαθήσαμε να φτιάξουμε μια απλή πλατφόρμα κοινωνικής δικτύωσης όπου οι χρήστες θα μπορούν να ανεβάζουν της φωτογραφίες τους και να σχολιάζουν φωτογραφίες άλλων χρηστών.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Ruby, Ruby on Rails, RoR, JavaScript, CSS, SASS, HTML, ReactJS, React Native, Progressive Web App, Responsive, Native, Web.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

ΕΙΣΑΓΩΓΗ

Η υλοποίηση της πτυχιακής έγινε σε τρία στάδια, αρχικά έπρεπε να σκεφτούμε την ιδέα και το πως θα μπορεί να υλοποιηθεί μια τέτοια εφαρμογή αλλά και με ποια εργαλεία θα ήταν πιο αποδοτικά και λειτουργικά, έτσι διαλέξαμε την Ruby on Rails γιατί έχουν δημιουργηθεί πολλές εφαρμογές πάνω σε αυτό το framework άρα είναι μια ασφαλής επιλογή αλλά και λόγω του μεγάλου community σε τυχών προβλήματα που θα αντιμετωπίζαμε κατά την υλοποίηση θα βρίσκαμε ποιο γρήγορα βοήθεια.

Προδιαγραφές της Εφαρμογής - Περιγραφή

Αρχικά σκεφτήκαμε για το τι εφαρμογή θα δημιουργήσουμε και καταλήξαμε να προσανατολιστούμε στα κοινωνικά δίκτυα και ονομάσαμε την εφαρμογή Moments.Στην εφαρμογή ο χρήστης έχει την δυνατότητα να δημιουργεί να προφίλ να μοιράζετε δεδομένα με τους φίλους του.

Σχεδιασμός

Ύστερα σχεδιάσαμε την εφαρμογή αρχικά στο χαρτί όπου δημιουργήσαμε της οντότητες της εφαρμογής και από ποιες σελίδες θα αποτελείτε αλλά και ένα πρωταρχικό design.Μετά στο illustrator σχεδιάσαμε ολόκληρή την εφαρμογή και όλα τις τα στάδια(Αρχική, Εγγραφή, Είσοδος, Τοίχος-Wall, Προφίλ κλπ.).

Υλοποίηση

Τέλος περάσαμε στο κομμάτι τις υλοποίησης όπου αρχικά φτιάξαμε το διάγραμμα ολόκληρης της εφαρμογής με τις οντότητες που έχει η εφαρμογή μας αλλά και την σχέση μεταξύ των οντοτήτων άρα σχεδιάσαμε την βάση μας και μετά περάσαμε στο κομμάτι του κώδικα και τις υλοποίησης.

Επιλέξαμε την Ruby μαζί με το framework Ruby on Rails για την υλοποίηση και διάφορα gems και στο front-end κομμάτι χρησιμοποιήσαμε καθαρή JS και CSS.

Στο 1ο Κεφάλαιο ασχοληθήκαμε με την ιστορία των social media το πότε άρχισαν, την εξέλιξη που είχαν με τα χρόνια, εταιρίες που χτίστηκαν πάνω σε αυτή την ιδέα και το πια ήταν τα κέρδη τους αλλά και την επανάσταση που έφεραν στο τρόπο που επικοινωνούν οι άνθρωποι μεταξύ τους, αλλά και οι αρνητικές επιπτώσεις που έχει αυτό στην κοινωνία.

Στο 2ο Κεφάλαιο ασχοληθήκαμε με την γλώσσα προγραμματισμού Ruby την ιστορία της, την δομή της αλλά και τις διαφορές που έχει σε σχέση με άλλες κλασσικές γλώσσες και τέλος το community γύρω από την Ruby.

Στο 3ο Κεφάλαιο ασχοληθήκαμε με το framework Ruby on Rails και την ιστορία του, την δομή ενός τυπικού MVC framework και το τι είναι αλλά και πώς λειτουργεί και τέλος τις εταιρίες που χρησιμοποιούν την Ruby on Rails σήμερα.

Στο 4ο Κεφάλαιο ασχοληθήκαμε με την γλώσσα προγραμματισμού JavaScript την ιστορία της και τους ιδιωματισμούς της αλλά και τα web-application και το πώς λειτουργούν στο οικοσύστημα της JavaScript.

Στο 5ο Κεφάλαιο ασχοληθήκαμε με την HTML και CSS την ιστορία τους και το πώς λειτουργούν σε μια ιστοσελίδα.

Στο 6ο Κεφάλαιο ασχοληθήκαμε με το Responsive Design το τι είναι και το πώς λειτουργεί, τις πιο νέες τεχνικές αλλά και τις διαφορές των web-application με τα mobile-application.

Στο 7ο Κεφάλαιο ασχοληθήκαμε με την εφαρμογή, την δομή της και αναλύσαμε κάθε κομμάτι της και το πώς λειτουργεί αλλά και το πως φτιάχτηκε με εκτενή αναφορά στον κώδικα και σχολιασμό στο τι κάνει κάθε κομμάτι του.

Στο 8ο Κεφάλαιο ασχοληθήκαμε με τα συμπεράσματα που είχαμε από την δημιουργία αυτής της εφαρμογής, οι δυσκολίες που αντιμετωπίσαμε αλλά και το τι μάθαμε από αυτό.

1 ΚΟΙΝΩΝΙΚΑ ΔΙΚΤΥΑ

1.1 Εισαγωγή

Οι ισότοποι κοινωνικής δικτύωσης γεννήθηκαν μέσα από το διαδίκτυο και την συνεχή εξέλιξη του όπου έφερε την ανάγκη στον κόσμο να δημιουργήσει καινούργια μέσα επικοινωνίας πιο άμεσα.

Τα παραδοσιακά μέσα μαζικής επικοινωνίας όπως είναι η τηλεόραση και το ραδιόφωνο δεν αρκούσαν πια για να καλύψουν τις ανάγκες του κόσμου, διότι δεν έδιναν την δυνατότητα στους ακροατές η τους τηλεθεατές να έρθουν σε επαφή και να ανταλλάξουν απόψεις αλλά και περιεχόμενο με άλλους ακροατές-τηλεθεατές.

1.2 Ιστορική Αναδρομή

Τα κοινωνικά δίκτυα έχουν μια μεγάλη ιστορία πού ξεκινάει από το 1971 με την αποστολή του πρώτου email μέχρι και σήμερα όπου έχουμε τα πιο σύγχρονα μέσα επικοινωνίας όπως το Snapchat οπού πλέον στέλνουμε μόνο βίντεο για να επικοινωνούμε μεταξύ μας.

Όλα αυτά τα χρόνια υπήρξε μια τρομερή εξέλιξη από το email ένα απλό κείμενο που άρχισε στέλνεται σε μεγάλους επιτραπέζιους υπολογιστές μέχρι και σήμερα όπου φτάσαμε στην εποχή του Instagram και του Snapchat οπού οι χρήστες επικοινωνούν μεταξύ τους μέσω εικόνων και βίντεο μέσα από μικρά και πολύ ελαφριά κινητά.

Για να φτάσουμε όμως στην σημερινή εποχή έχουμε περάσει από πολλά διαφορετικά στάδια και από πολλές κοινωνικές πλατφόρμες. Αρχικά ξεκινήσαμε με το Friendster και το Myspace οπού είχαν αρκετούς εκατομμύρια χρήστες όπου επικοινωνούσαν και μοιράζονταν περιεχόμενο μεταξύ τους.

1.3 Παραδείγματα Ισοτόπων

Ύστερα όμως ήρθε η εποχή του Facebook το 2004 όπου από ένα απλό και μικρό κοινωνικό δίκτυο που ξεκίνησε για τους φοιτητές του Harvard και μόνο εξελίχθηκε στην σημερινή του μορφή να αριθμεί περίπου
2 δισεκατομμύρια μηνιαίους χρήστες ένας αριθμός αστρονομικός, αυτό δείχνει την ανάγκη των ανθρώπων για επικοινωνία και ανταλλαγή απόψεων και ιδεών. Όμως δεν φτάνει μόνο αυτό αλλά χρειάστηκε και μια άψογή υλοποίηση και συντήρηση από τους ιδρυτές του ώστε να αυξάνει

τους χρήστες του κάθε χρόνο. Πράγμα πολύ δύσκολο να γίνει για οποιαδήποτε εταιρία στον κόσμο.

Τέλος να φτάσουμε στην σημερινή εποχή όπου έχουν φτιαχτεί πιο εξειδικευμένες πλατφόρμες για την επικοινωνία μας, όπως είναι το Instagram οπού ο κόσμος μοιράζεται τις στιγμές του μέσα από φωτογραφίες και βίντεο, το Tinder μια εφαρμογή όπου έχει σκοπό την γνωριμία μεταξύ δυο ανθρώπων αλλά και το GitHub οπού είναι ένα διαφορετικό μοντέλο κοινωνικού δικτύου διότι είναι μόνο για προγραμματιστές και μοιράζονται τον κώδικά τους αλλά και ιδέες και απόψεις πάνω σε συγκεκριμένα θέματα και προβλήματα.

1.4 Πλεονεκτήματα - Μειονεκτήματα

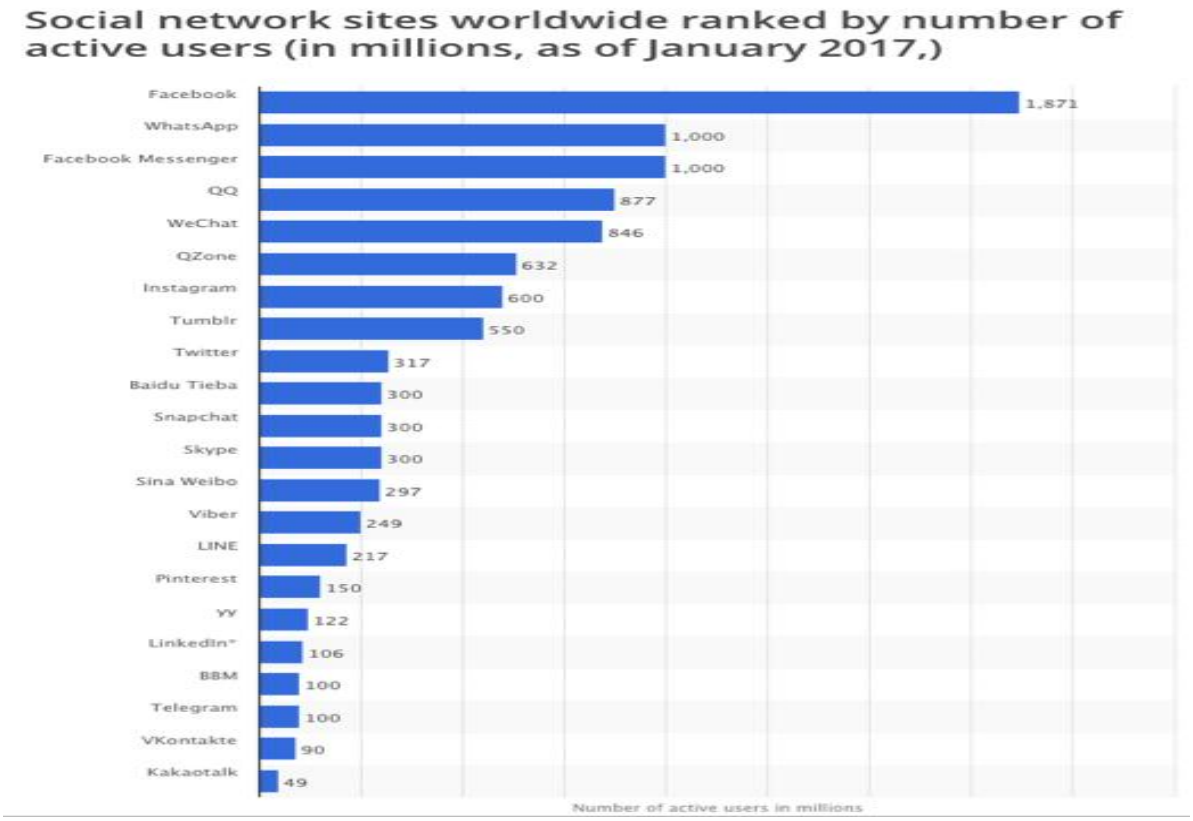
Αυτό το κενό ήρθαν να καλύψουν τα κοινωνικά δίκτυα δίνοντας την δυνατότητα στους χρήστες του να επικοινωνούν άνθρωποι από όλο τον κόσμο διαφορετικής θρησκείας, χρώματος, χώρας και δίνοντας την δυνατότητα σε αυτούς τους ανθρώπους να μοιράζονται περιεχόμενο(φωτογραφίες, μουσική κλπ.) με τα αγαπημένα τους πρόσωπα αλλά και με νέο κόσμο ανοίγοντας τον κύκλο τους και γνωρίζοντας καινούργιες ιδέες και πολιτισμούς.

Μέσα από τα κοινωνικά δίκτυα άνθρωποι ήρθαν πιο κοντά και άρχισαν να επικοινωνούν σε καθημερνή βάση με τα οικεία τους πρόσωπα κάτι που δεν μπορούσαν να κάνουν παλιότερα με τα παραδοσιακά μέσα επικοινωνίας.

Τα κοινωνικά δίκτυα όμως δεν είχαν μόνο θετικό αντίκτυπο στην ζωή των ανθρώπων εμφάνισε και μια αρνητική πλευρά που εν τέλει αποδείχτηκε πολύ επικίνδυνη, αυτή των ψευδών ειδήσεων όπου διαδίδονταν μέσα από τα κοινωνικά δίκτυα, αυτό είχε ως αποτέλεσμα την παραπληροφόρηση και την πλήρη αλλοίωση της αλήθειας.

Αυτό συνέβη διότι ειδήσεις μέσα από τα κοινωνικά δίκτυα δεν φιλτράρονται από κάπου αλλά ο καθένας μπορεί ανώνυμα να παραθέσει την δικιά του αλήθεια χωρίς στοιχεία.

Αυτό είχε ως αποτέλεσμα και ένα πρόσφατο γεγονός στις αμερικανικές εκλογές ένα κομμάτι των ψηφοφόρων ψήφισε για πρόεδρο της Αμερικής τον Donald Trump βασιζόμενη σε ψευδές ειδήσεις που διέρρεαν στο Facebook.



Εικόνα 1.1 Ενεργοί χρήστες στα κοινωνικά δίκτυα.
(http://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research/)

1.5 Επιχειρηματικότητα και κέρδη μέσω των κοινωνικών δικτύων

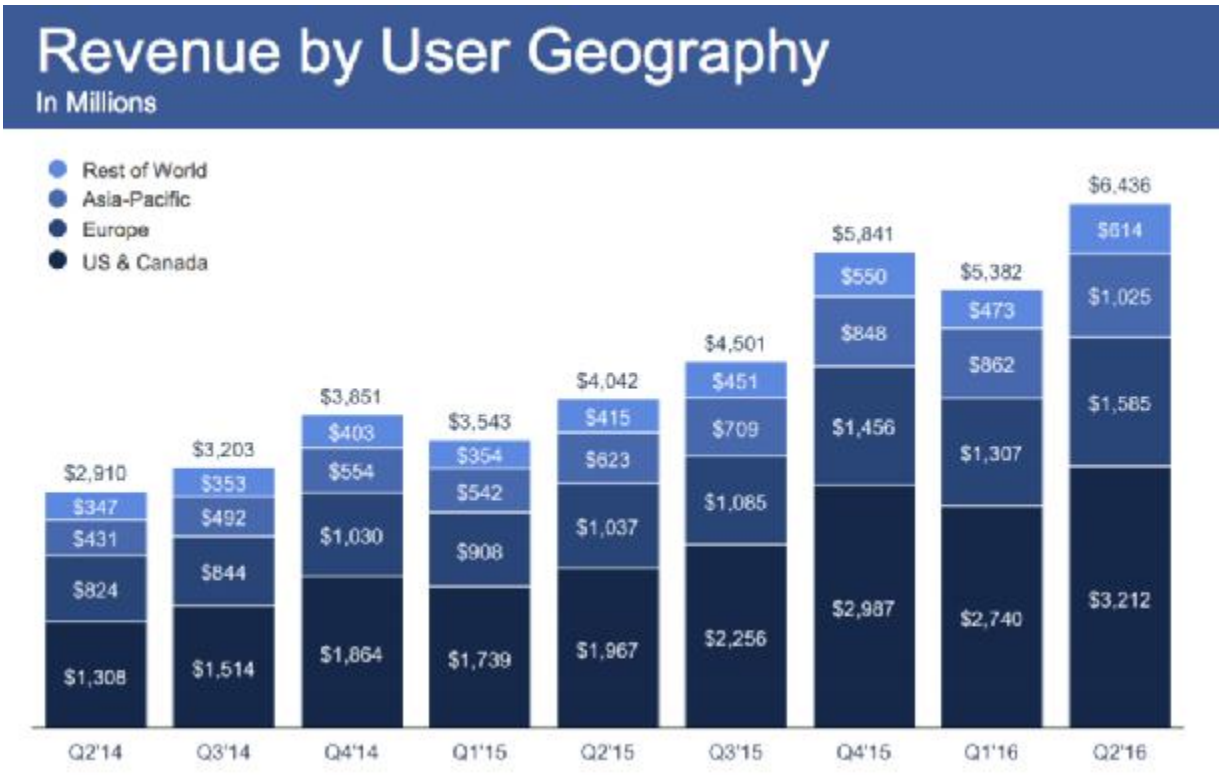
Στα προηγούμενα χρόνια η επένδυση σε ένα μέσο κοινωνικής δικτύωσης δεν θα ήταν καθόλου επικερδής σύμφωνα με τους επενδυτές και θα προτιμούσαν να επενδύσουν σε ένα πιο ασφαλές πεδίο και σίγουρο(πχ. ναυτιλία) και πιο ασφαλές.

Αλλά αυτό έχει αλλάξει ριζικά τα τελευταία χρόνια με κολοσσούς της τεχνολογίας όπως το Facebook, LinkedIn, Twitter κλπ., να έχουν κέρδη αστρονομικά ποσά και να επεκτείνουν τις επιχειρήσεις τους σε όλο τον κόσμο, Για παράδειγμα το Facebook αριθμεί πάνω από 7.000 εργαζομένους.

Αυτό βέβαια δεν είναι εύκολο και χρειάζεται χρόνια προσπάθειας και επένδυσης ενός κράτους σε τομείς υψηλής τεχνολογίας, διότι δεν είναι τυχαίο ότι όλες οι εταιρίες υψηλής τεχνολογίας εδρεύουν στο San Francisco (Silicon Valley
Επειδή σε εκείνη την πόλη εκτός από την φιλοσοφία που υπάρχει για την νεοφυή επιχειρηματικότητα και τις ευκαιρίες που πρέπει να δίνονται σε νέα παιδία να καινοτομούν, υπάρχει και ένα κράτος φιλικό προς τις επιχειρήσεις με μικρή γραφειοκρατία που σε διευκολύνει να αρχίσεις κάτι καινούργιο.

Αυτό σε συνδυασμό με το Stanford το καλύτερο πανεπιστήμιο στον κόσμο σε θέματα πληροφορικής όπου βγάζει εξαίρετους μηχανικούς και σε συνδυασμό με τους επενδυτές(Angel Investors) που δίνουν την ευκαιρία σε νέα παιδιά να κάνουν το όνειρο τους πραγματικότητα.

Όμως υπάρχει και μια άλλη πλευρά της πραγματικότητας όπου δείχνει ότι οι περισσότερες από τις νεοφυής επιχειρήσεις τείνουν να αποτυγχάνουν σε ένα βραχυπρόθεσμο χρόνο διότι η αγορά αλλάζει συνεχώς και μπορεί μια εταιρία μικρή να μην αντέξει αυτήν την γρήγορη αλλαγή αλλά αυτό δεν θα έπρεπε να αποθαρρύνει κανέναν διότι υπάρχουν και λαμπρά παραδείγματα απλά πρέπει να είναι πιο προσεκτικοί και σίγουροι όταν πάνε να ξεκινήσουν κάτι νέο.



Εικόνα 1.2 Κέρδη του Facebook σε όλο τον κόσμο.
(http://www.businessinsider.com/facebook-q2-2016-earnings-2016-7)

2 Η ΓΛΩΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ RUBY

Η Ruby είναι μια δυναμική γλώσσα προγραμματισμού με περίπλοκη αλλά εκφραστική γραμματική και σημαντική βιβλιοθήκη κλάσεων ενσωματωμένα σε ένα ευρύ και δυνατό API. Η Ruby έχει βασιστεί στην Lisp, Smalltalk και Perl, αλλά χρησιμοποιεί γραμματική που είναι εύκολη στην κατανόηση για προγραμματιστές C και Java.

Η Ruby είναι μια καθαρά αντικειμενοστραφής γλώσσα προγραμματισμού αλλά χρησιμοποιείται και με διαδικαστικό ή λειτουργικό στυλ. Έχει καλές δυνατότητες μετά-προγραμματισμού και μπορεί ως αποτέλεσμα να χρησιμοποιηθεί στη δημιουργία γλώσσας συγκεκριμένα για ένα τομέα (domain specific language).

Επίσης είναι δομημένη σε functions κάτι που την κάνει απλή και κατανοητή και όταν ένα project μεγαλώνει και προστίθενται συνέχεια καινούργια στοιχεία είναι πολύ πιο διαχειρίσιμο το testing όταν γίνεται σε κομμάτια.

Επιπλέον είναι ιδανική γλώσσα για αρχάριους διότι είναι εύκολη στο συντακτικό και δεν σε μπερδεύει με περίπλοκους όρους, αλλά μαθαίνει τα βασικά του προγραμματισμού με απλό και κατανοητό τρόπο.

Η Ruby χρησιμοποιείται κυρίως για την επικοινωνία με τους servers για αυτό και ονομάζεται server-side language και σε συνδυασμό με την Ruby on Rails είναι το ιδανική γλώσσα προγραμματισμού για να φτιάξεις web-applications.

Επιπλέον όταν ο δημιουργός της την έφτιαξε μόνος του δεν φανταζόταν το πόσο μεγάλη γλώσσα θα γινόταν και το τεράστιο community που θα αποκτούσε λόγω της open-source κουλτούρας και να αριθμεί πάνω από 40 contributors.

Μεγάλο πλεονέκτημα της Ruby είναι και το μεγάλο community που έχει με αποτέλεσμα σε οποιοδήποτε πρόβλημα και αν παρουσιαστεί να βρεθεί η λύση μέσα από το community διότι 99% θα το έχει αντιμετωπίσει κάποιος άλλος πιο πριν.

2.1 Η Ιστορία της Ruby

Yukihiro “Matz” Matsumoto (Δημιουργούς της Ruby)

"Γνώριζα πολλές γλώσσες πριν δημιουργήσω τη Ruby, αλλά ποτέ δεν ήμουν ευχαριστημένος με αυτές. Ήταν άσχημες, δύσκολες και πιο περίπλοκες ή πιο απλές από'ότι περίμενα.

Ήθελα να δημιουργήσω τη δική μου γλώσσα που να με καλύπτει σαν προγραμματιστή. Ήξερα πολλά για το κοινό στο οποίο απευθυνόταν αυτή η γλώσσα. Προς έκπληξη μου, πολλοί προγραμματιστές ανά τον κόσμο ένιωθαν όπως εγώ. Νιώθουν χαρούμενοι όταν ανακαλύπτουν και

γράφουν κώδικα στη Ruby. Καθ''όλη τη διάρκεια σχεδιασμού και υλοποίησης της Ruby έβαλα όλη την ενέργεια και το κίνητρο μου προς το σκοπό να γίνει απλή και γρήγορη.

Οι περισσότεροι προγραμματιστές πιστεύουν ότι είναι κομψό, εύκολο στη χρήση και απόλαυση να γράφουν κώδικα με Ruby. Η φιλοσοφία του Matz για το σχεδιασμό της Ruby περιληπτικά είναι μια έκφραση που χρησιμοποιεί συχνά: Η Ruby σχεδιάστηκε για να κάνει τους προγραμματιστές χαρούμενους."

Η Ruby γεννήθηκε το 1993, η πρώτη ιδέα της οποίας συνελήφθη σε μια συζήτηση του Yukihiro Matsumoto ("Matz") με ένα συνάδελφό του. Συζητούσαν την πιθανότητα δημιουργίας μιας αντικειμενοστραφούς γλώσσας προγραμματισμού.

Ο Matz είπε σε μια συνέντευξη ότι ήξερε Perl αλλά ότι δεν του άρεσε για αυτό φαινόταν σαν παιχνίδι. Είπε επίσης ότι ήξερε Python αλλά δεν του άρεσε γιατί δεν ήταν πραγματική γλώσσα αντικειμενοστραφούς προγραμματισμού.

Το 1998 ο Matz έφτιαξε μια απλή σελίδα στα αγγλικά για τη Ruby. Η Ruby όμως τότε ήταν ακόμα δημοφιλής μόνο τοπικά στην Ιαπωνία. Σε μια προσπάθεια να προχωρήσει η επέκταση της Ruby έφτιαξε το Ruby-talk. Έτσι ξεκίνησε η επέκταση της Ruby και εκτός Ιαπωνίας.

2.2 Το συντακτικό της Ruby

Η Ruby σε σύγκριση με άλλες γλώσσες έχει πολλές διαφορές διότι δεν είναι απαραίτητο να χρησιμοποιούνται παρενθέσεις η άγκιστρα για να τρέξει ο κώδικας αλλά είναι πιο απλή και κατανοητή.

Επιπλέον μια άλλη κύρια διαφορά σε σχέση με άλλες γλώσσες(C, C++) είναι οι μέθοδοι που χρησιμοποιεί η Ruby και απλουστεύουν πιο πολύ τον κώδικά.

Πχ. μια απλή function στην C++ για την προσθήκη των στοιχείων ενός πίνακα έχει αυτήν την δομή όπου είναι αρκετά πολύπλοκη για κάποιον αρχάριο:

```
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

int multiply(int);

int main() {
    vector<int> source;
    for(int i = 1; i <= 5; i++) {
    source.push_back(i);
```

```
}
   vector<int> result;
    result.resize(source.size());
    transform(source.begin(), source.end(), result.begin(), multiply);

    for(vector<int>::iterator it = result.begin(); it != result.end(); ++it) {
        cout << *it << endl;
    }
}

int multiply(int value) {
    return value * 10;
}
```

Ενώ στην Ruby *έχει αυτήν την δομή όπου είναι πολύ πιο απλή και γράφεται σε μια γραμμή κώδικα:*

```
[1, 2, 3].map { |n| n * n } #=> [1, 4, 9]
```

Εδώ βλέπουμε την απλότητα του κώδικα όπου χρησιμοποιώντας μια απλή μέθοδο map *έχουμε την δυνατότητα να επεξεργαστούμε τον αρχικό μας πίνακα και να πάρουμε έναν επεξεργαζόμενο πίνακα με νέα στοιχεία κάτι που το καταφέρνουμε με μια γραμμή κώδικα.*

3 ΤΟ FRAMEWORK ΤΗΣ RUBY ON RAILS

Αρχικά οι περισσότεροι προγραμματιστές που έφτιαχναν εφαρμογές για το διαδίκτυο σε οποιαδήποτε γλώσσα και αν χρησιμοποιούσαν(JAVA, .NET, PHP) και σε οποιοδήποτε framework και αν βασιζόντουσαν δεν ήταν απόλυτα ευχαριστημένοι με το αποτέλεσμα που είχαν.

Τότε εμφανίστηκε στο προσκήνιο των web applications η Ruby on Rails και άλλαξε τα δεδομένα που υπήρχαν μέχρι τότε στον σχεδιασμό και στην υλοποίηση μια web εφαρμογής. Η Ruby on Rails για τον σχεδιασμό και την υλοποίηση της εφαρμογής χρησιμοποιεί το Model-View-Controller αν και στην Java με το Tapestry framework χρησιμοποιούσε το MVC η Ruby on Rails άλλαξε εντελώς τον τρόπο σχεδιασμού της εφαρμογής.

Επίσης σε οποιαδήποτε γλώσσα και σε οποιοδήποτε framework και αν χρησιμοποιούν οι προγραμματιστές για την υλοποίηση της εφαρμογής ένα μεγάλο κομμάτι είναι το testing όπου είναι πολύ σημαντικό για μια μεγάλη εφαρμογή για τα bugs που μπορεί ναι προκύψουν. Αλλά η Ruby on Rails δίνει ένα μεγάλο βάρος στο testing και είναι σχεδιασμένη έτσι ώστε πριν από την υλοποίηση του κώδικα π.χ. μια function που είναι υπεύθυνη για μια συμπεριφορά στην εφαρμογή να γράφεται πρώτα το αντίστοιχο test της αντίστοιχής function.

Οι εφαρμογές σε Ruby on Rails γράφονται στην Ruby μια μοντέρνα γλώσσα οπού είναι εύκολη στην υλοποίηση και μπορεί να επεξεργαστεί μετά από πολύ καιρό ο κώδικας της εφαρμογής κάτι που είναι πολύ σημαντικό για την επέκταση της εφαρμογής.

Ένα μικρό παράδειγμα κώδικα μια Ruby on Rails εφαρμογής είναι το παρακάτω:

```
class Project < ActiveRecord::Base
  belongs_to :portfolio
  has_one :project_manager
  has_many :milestones
  has_many   :deliverables, :through => :milestones

  validates  :name, :description, :presence => true
  validates  :non_disclosure_agreement, :acceptance => true
  validates  :short_name, :uniqueness => true
end
```

Σε αυτό το μικρό παράδειγμα βλέπουμε πολλές από της λειτουργίες της Ruby on Rails όπως της κλάσεις και την κληρονομικότητα και την σχέσεις που έχουν μεταξύ τους οι κλάσεις μεταξύ τους.
Πιο αναλυτικά έχουμε την κλάση μας Project οπού κληρονομεί τα στοιχεία της από την ActiveRecord οπού είναι το M από το MVC δηλαδή το βασικό μοντέλο που δουλεύομε στην Rails.
Υστέρα βλέπουμε μέσα από κάποιες λέξεις κλειδιά δημιουργούμε τις σχέσεις που έχουν μεταξύ τους οι κλάσεις μας (belogns_to, has_one, has_many).
Μετά μέσα από την λέξη κλειδί validates κάνουμε τους απαραίτητους ελέγχους δηλαδή το μέγεθος ενός username η password την μοναδικότητα κλπ.

3.1 Η Ιστορία της Ruby on Rails

Η Ruby on Rails δημιουργήθηκε το 2003 από τον David Heinemeier Hansson καθώς δούλευε σε ένα νέο project της Basecamp οπού είναι ένα εργαλείο διαχείρισης των projects από την 37signals.Η επίσημη έκδοση της Ruby on Rails σε ανοιχτού κώδικα project έγινε τον Ιούλιο του 2004.

Ο David Heinemeier Hansson έχει πει ότι η δημιουργία αυτού του νέου framework ήταν κάτι πολύ απαιτητικό και δύσκολο και ότι αντιμετώπισε πολλές δυσκολίες, ο λόγος της υλοποίησης ήταν ότι τα μέχρι τότε εργαλεία που υπήρχαν στην αγορά είτε κλειστού η ανοιχτού κώδικα δεν κάλυπταν τις ανάγκες του που είχε ως προγραμματιστής και στο project που είχε αναλάβει από την 37signals.

Επίσης ξεκαθάρισε ότι αν είχε ξεκινήσει το ίδιο project δηλαδή την δημιουργία της Ruby on Rails σε κάποια άλλη γλώσσα εκτός της Ruby μάλλον δεν θα το τελείωνε ποτέ διότι η Ruby είναι μια βατή γλώσσα με εύκολες έννοιες και καθόλου δυσνόητη κάτι που διευκολύνει πολύ την δουλεία ενός προγραμματιστή να συγκεντρωθεί στα απολύτως απαραίτητα.

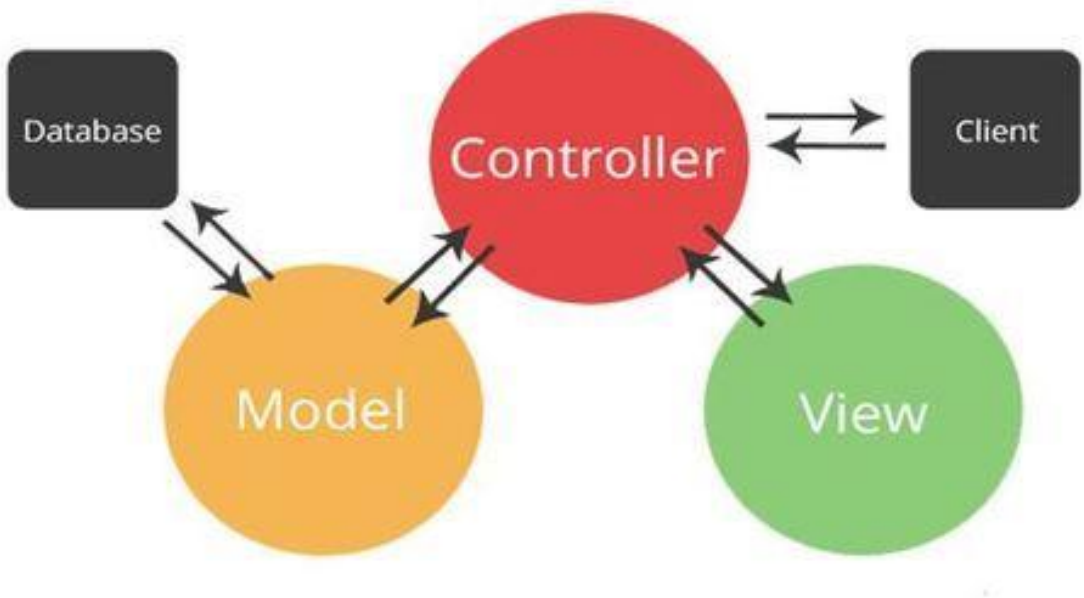
Η Ruby on Rails έχει κάποιες βασικές αρχές στις όποιες είναι δομημένη, αρχικά ακολουθεί το Model-View-Controller πρότυπο για την δημιουργία των web applications. Επίσης το Active Record οπού είναι ένα σχεδιαστικό μοντέλο για την αποθήκευση των δεδομένων στην βάση και με ποιόν τρόπο θα υλοποιείται, χρησιμοποιώντας το CRUD(Create-Read-Update-Delete).

Πιο αναλυτικά το CRUD είναι μια αρχή για να δημιουργούμε εφαρμογές όπου έχει τις βασικές λειτουργίες της εφαρμογής C (create) που είναι η δημιουργία πχ ενός post από τον χρήστη, R (read) είναι οι πληροφορίες που δίνονται στον χρήστη από την εφαρμογή πχ τα υπόλοιπα posts από τους χρήστες, U (update) είναι η ανανέωση των στοιχείων πχ η αλλαγή των προσωπικών

δεδομένων από τον χρήστη και τέλος το D (delete) όπου είναι η διαγραφή των δεδομένων από τον χρήστη πχ ένα post που δεν του άρεσε.

Μια άλλη βασική αρχή είναι το DRY(Don’t Repeat Yourself) δηλαδή η αρχή του να μην επαναλαμβάνεις άσκοπα των κώδικα σε πολλά μέρη του project οπού δεν χρειάζεται αλλά να χωρίζεις των κώδικα κομμάτια(components) οπού μπορούν να λειτουργούν και αυτόνομα έτσι ώστε να έχει μια σωστή δομή ο κώδικας αυτό έχει πολλά θετικά, αρχικά ότι είναι πιο διαχειρίσιμο το project στο κομμάτι του testing αλλά και ότι είναι πιο ευανάγνωστός ο κώδικας από άλλους προγραμματιστές.

3.2 Η αρχιτεκτονική MVC (Models-Views-Controllers)



Εικόνα 1.3 MVC Μοτίβο
(https://www.indiamart.com/proddetail/mvc-framework-development-service-14325864648.html)

Στο παραπάνω διάγραμμα βλέπουμε το μοντέλο MVC που είναι ο κορμός της ruby on rails, ξεκινώντας με το Model που είναι το Object μας δηλαδή μέσα του είναι όλα τα δεδομένα της εφαρμογής μας και όπως βλέπουμε στο διάγραμμα έρχεται σε επικοινωνία με τους controllers για να δώσει και να πάρει δεδομένα.

Το View είναι αυτό που βλέπει ο χρήστης με αυτό που έρχεται σε επαφή και αλληλοεπιδρά και όπως βλέπουμε και στο διάγραμμα έρχεται σε επικοινωνία με τους controller για να στείλει τις νέες ενέργειες του χρήστη αλλά και να κατευθυνθεί από τους controllers στο τι θα κάνει.

Οι Controllers θα ανταποκριθούν σε διάφορες ενέργειες του χρήστη και θα κάνουν αλλαγές και στο Model αλλά και στο View όπως βλέπουμε και στο διάγραμμα.

Η Rails είναι ένα framework για την δημιουργία web εφαρμογών σύμφωνα με το Model-View-Controller πρότυπο, αν κατανοήσουμε πως λειτουργεί το MVC πρότυπο τότε θα κατανοήσουμε πως λειτουργεί και η Ruby on Rails αλλά και αλλά αντίστοιχα framework.

Το Model από το M-V-C αντιπροσωπεύει το μοντέλο δηλαδή τα δεδομένα μας και το πώς αποθηκεύονται αλλά και το που, δηλαδή είναι η βάση δεδομένων μας όπου υποστηρίζεται από τις κατηγορίες μοντέλων που προέρχονται από την Active Record :: Base.Η Active Record διαχειρίζεται τα δεδομένα μας και μας επιτρέπει την σύνδεση με την λογική μας μέσω των Controllers.

Το View από το M-V-C αντιπροσωπεύει και είναι υπεύθυνο για την εμφάνιση της εφαρμογής μας και η αλληλεπίδραση που θα έχει ο χρήστης με την εφαρμογή μας και την μετέπειτα συμπεριφορά της εφαρμογής. Είναι δομημένα με HTML κώδικα με ενσωματωμένο κώδικα Ruby (αρχεία ERB).

Τα Views εμφανίζουν την αντίδραση από τα Controllers. Επίσης μπορεί να δημιουργήσει το σώμα ενός μηνύματος ηλεκτρονικού ταχυδρομείου. Στην Rails, τα Views εξαρτιόνται από το Action View.

Το Controller από το M-V-C αντιπροσωπεύει την λογική μας δηλαδή το πώς θα συμπεριφέρεται η εφαρμογή μας και η διαχείριση της βάσης μας ώστε να παίρνουμε τα δεδομένα μας και να τα χειριζόμαστε αλλά και να τα επεξεργαζόμαστε. Άρα με λίγα λόγια οι Controllers φορτώνουν και χειρίζονται τα Models.

Το MVC πρότυπο είναι ένα πολύ σημαντικό κομμάτι στην ιστορία των web εφαρμογών και σε οποιαδήποτε γλώσσα(Ruby, PHP, JavaScript) ακολουθεί την ίδια λογική και δομή.

3.3 Εταιρίες που χρησιμοποιούν την Ruby on Rails σήμερα

Η Ruby on Rails είναι ένα ανοιχτού κώδικα framework αλλά βλέπουμε ότι χρησιμοποιείται από πάρα πολλές εταιρίες που είναι κορυφαίες στον κόσμο της τεχνολογίας όπως είναι η GitHub, Shopify, Airbnb, Twitch, Sound Cloud, Hulu, Zen desk και από πολλές ακόμα ανερχόμενες εταιρίες.

Αυτό οφείλετε στις λειτουργίες που προσφέρει το ίδιο το framework και στην πιο αποτελεσματική λειτουργία του από κάποιο ενδοεταιρικό(custom) framework κλειστού κώδικα αλλά και στο τεράστιο community οπού ανανεώνει και διορθώνει την Rails.

Βέβαια αυτό το φαινόμενο να χρησιμοποιούν εταιρίες ανοιχτού κώδικα project, να συνεισφέρουν σε αυτά αλλά και να δημιουργούν εργαλεία και να τα ανεβάσουν υπό ανοιχτού

κώδικα άδεια είναι πρόσφατο. Διότι στο παρελθόν οι εταιρίες χρησιμοποιούσαν μόνο κλειστού κώδικα εργαλεία και δεν μοιράζονταν τις γνώσεις τους η κάποια λειτουργεία που κατάφεραν να φτιάξουν.

Σε όλο αυτό έχει βοηθήσει η ανοιχτού κώδικα φιλοσοφία που έχει αρχίσει και επικρατεί τα τελευταία χρόνια και έχει ως αποτέλεσμα να βλέπουμε την συνεργασία εταιριών σε ανοιχτού κώδικα εργαλεία και frameworks και να συνεισφέρουν όλες μαζί σε αυτό.

Με αποτέλεσμα να έχουν δημιουργήσει απίστευτα πράγματα όπως είναι η Ruby on Rails οπού από πίσω είναι η Basecamp και στο οικοσύστημα της JavaScript έχουν φτιαχτεί απίστευτα libraries και framework όπου έφεραν επανάσταση στον τρόπο που σκεφτόμασταν τα τελευταία χρόνια τον σχεδιασμό και την υλοποίηση των web εφαρμογών.

Κάποια από αυτά τα framework είναι η ReactJS το όποιο είναι ένα library που έχει δημιουργήσει το Facebook για front-end engineers και έχει εισάγει στο κόσμο των web εφαρμογών νέες έννοιες και νέες λειτουργίες όπως το Virtual DOM και τα Components τεχνολογείς πολύ πιο αποτελεσματικές και πιο έξυπνες όπου βλέπουμε να τις εισάγουν και άλλα libraries όπως είναι η AngularJS ένα framework της Google για δημιουργία web εφαρμογών αλλά και πολλές άλλες εταιρίες όπως το Twitter, Airbnb κλπ.

\

4 Η ΓΛΩΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ JAVASCRIPT

Η JavaScript είναι μια πολύ σημαντική γλώσσα διότι είναι η γλώσσα που επικοινωνεί με τους φυλλομετρητές και είναι αυτό που την κάνει από τις πιο γνωστές και χρήσιμες γλώσσες προγραμματισμού.

Από την άλλη πλευρά είναι η πιο παρεξηγημένη γλώσσα προγραμματισμού διότι πολύ την φέρουν δύσκολη η 'άσχημη' γλώσσα αλλά ουσιαστικά για αυτό ευθύνεται το API του browser το DOM(Document Object Model). Αλλά το DOM είναι δύσκολο να το διαχειριστείς σε οποιαδήποτε γλώσσα και αν χρησιμοποιήσεις με αποτέλεσμα να κάνει την δημιουργία web εφαρμογών αρκετά δύσκολη.

Με αποτέλεσμα να δημιουργηθούν όλα αυτά τα libraries και τα framework(Angular, React, Ember etc..) που έχουμε τα τελευταία χρόνια και κάνουν την δουλειά των προγραμματιστών αρκετά πιο εύκολη.

Το φοβερό με την JavaScript είναι ότι μπορεί να ξεκινήσει κάποιος να γράφει κώδικα και μικρές web εφαρμογές χωρίς να ξέρει και πολλά για την ίδια την γλώσσα αλλά και γενικά από προγραμματισμό, Είναι μία γλώσσα με ατελείωτες δυνατότητες.

Η JavaScript όπως και όλες οι γλώσσες προγραμματισμού έχουν καλά και κακά κομμάτια απλά τα κακά κομμάτια της JS τα αντιμετωπίζουμε καθημερνά διότι όλοι ερχόμαστε σε επαφή με τους browsers, αλλά η JavaScript έχει μια πολύ μεγάλη κοινότητα όπου φροντίζει να διορθώνει λάθη αλλά και να προσθέτει λειτουργίες από άλλες γλώσσες που ταιριάζουν στην φιλοσοφία της JavaScript.

Με αποτέλεσμα στην τελευταία έκδοση της ECMAScript 6 (ES6) να έχει διορθώσει πολλά λάθη που ταλαιπωρούσαν τους προγραμματιστές για χρόνια(πχ scope, classes etc..) αλλά και να έχουν προστεθεί νέες δυνατότητες στην γλώσσα που την κάνουν ακόμα καλύτερη.

"Η JavaScript είναι η μόνη γλώσσα που φοβάμαι ότι ο κόσμος νιώθει ότι δεν χρειάζεται να την μάθει πριν την χρησιμοποιήσει."

Douglas Crockford

Όλα συνέβησαν σε έξι μήνες από τον Μάιο μέχρι τον Δεκέμβριο του 1995. Η Netscape είχε μια ισχυρή παρουσία στον κόσμο της πληροφορικής, Ο Marc Andreesen ιδιοκτήτης της Netscape είχε ένα όραμα ότι το διαδίκτυο θα έπρεπε να γίνει πιο δια δραστικό animation και άλλες πιο αυτόματες διαδικασίες.

Άρα γεννήθηκε η ανάγκη για την δημιουργία μιας γλώσσας που θα επικοινωνεί με το DOM.Εκείνη την εποχή η Java ήταν σε άνθηση και άρχιζαν να φτιάχνομαι η πρώτες εφαρμογές

σε Java.Αλλά η γλώσσα που θα έπρεπε να δημιουργηθεί θα έπρεπε να είναι διαφορετική και να καλύπτει
διαφορετικό κοινό και ανάγκες από την Java.

Η HTML ήταν στις αρχές τις και ήταν εύκολη και προσβάσιμη και από μη προγραμματιστές άρα και η γλώσσα που θα έπρεπε να δημιουργήσει η Netscape θα έπρεπε να είναι απλή άλλα συγχρόνως με πολλές δυνατότητες.

Τότε ήταν που ο Brendan Eich ο πατέρας της JavaScript εμφανίστηκε στο προσκήνιο. Ο Eich είχε υπογράψει μια σύμβαση ώστε να δημιουργήσει μια γλώσσα για τον browser.

Εκείνη την περίοδο η Java είχε αρχίσει να τραβάει το ενδιαφέρον του κόσμου και η Sun Microsystems ήταν σε συμφωνά με την Netscape για να κάνει την Java προσβάσιμη και στους browser.Και τότε ήρθε η ιδέα να την ονομάσουν Mocha σκέφτηκαν μια διαφορετική ονομασία διότι θα απευθυνόταν σε διαφορετικό κοινό από την Java.

Υπήρχε μεγάλη πίεση τότε στον Eich να δημιουργήσει μια γλώσσα γρήγορα γιατί ο ανταγωνισμός ήταν μεγάλος είχε ένα πλεονέκτημα να επιλέξει όποια εργαλεία ήθελε αλλά και ένα μεγάλο μειονέκτημα τον χρόνο.

Η πρωτότυπη έκδοση της Mocha ήταν έτοιμη τον Μάιο του 1995 σε μικρό χρονικό διάστημα μετονομάστηκε σε Live Script και τον Δεκέμβριο η Sun με την Netscape έκλεισαν την συμφωνία με την ονομασία Mocha/Live Script όπου θα μετονομαζόταν σε JavaScript.

Δεν ξέρουμε τι θα συνέβαινε αν ο Eich είχε αποτύχει να τελειώσει την παράδοση του project στον συγκεκριμένο χρόνο, οι εναλλακτικές που υπήρχαν(Python, Tcl, Scheme) δεν είχαν καμία σχέση με την Java.

Όταν η Sun και η Netscape αποφάσισαν να μετονομάσουν την γλώσσα από Mocha/Live Script σε JavaScript και εκείνο τον καιρό η Netscape είχε αρχίσει να γίνεται ο αγαπημένος browser του κόσμου και η Microsoft είχε αρχίσει να φτιάχνει τον Internet Explorer.

Από τις πρώτες μέρες η JavaScript είχε δημιουργήσει θετικό κλίμα και είχε αλλάξει το δεδομένα που υπήρχαν μέχρι τότε στην διεπαφή του χρήστη με τον browser και η Microsoft λόγω του ανταγωνισμού είχε αρχίσει να φτιάχνει την δική της JavaScript με την ονομασία JScript. Η JScript εκτός από διαφορετικό όνομα είχαν και πολύ μεγάλες διαφορές με την JavaScript.

Στα τέλη του 1996 ο Eich ξαναέγραψε από την αρχή την Mocha(JavaScript) σε μια πιο καθαρή μορφή της. Αυτή η νέα μηχανή της Netscape για την JavaScript ονομάζεται Spider Monkey.

Αλλά η JavaScript επειδή γεννήθηκε γρήγορα πολλές από τις δυνατότητες της δεν ήταν από την αρχή αυτές η δυνατότητες της γλώσσας την καθόρισαν ως γλώσσα.

Ωστόσο το να έχει η JavaScript με την Java παρεμφερείς κώδικα δεν ήταν από την αρχή στους στόχους αλλά λόγω του marketing αυτό άλλαξε.

5 Η Ιστορία της HTML και CSS

Όλες οι ιστοσελίδες χρησιμοποιούν HTML και CSS για την βασική δομή τους και ανάλογα με τις απαιτήσεις και την πολυπλοκότητα της κάθε σελίδας προστίθενται και κάποιες άλλες τεχνολογίες.

Οι περισσότερες ιστοσελίδες επίσης περιλαμβάνουν και επιπλέον περιεχόμενο όπως εικόνες, ήχο, βίντεο, animation.Κάποιες άλλες σελίδες στέλνουν JavaScript ή Flash στους browsers μας όταν πρόκειται για κάτι δια δραστικό περιεχόμενο που απαιτεί λογική από πίσω χρειάζεται μια γλώσσα προγραμματισμού.

Πολλές μικρές ιστοσελίδες χρησιμοποιούν μόνο HTML και CSS αλλά μεγαλύτερες που χρειάζονται συνεχή ανανέωση στο περιεχόμενο χρησιμοποιούν κάποιο είδους CMS(content management system) όπως WordPress, Joomla η κάποιο custom CMS που υλοποιείται για τις ανάγκες τις συγκεκριμένης εταιρίας.

Επίσης αυτές οι μεγάλες ιστοσελίδες λόγω τις συνεχής ανανέωσης του περιεχομένου χρειάζονται και μια βάση δεδομένων για να αποθηκεύονται τα δεδομένα τους και γλώσσες όπως PHP, Java, Ruby, Python.

Άρα η HMTL είναι η δομή μιας ιστοσελίδας και μέσα από τα tags της HTML(, <header>, <footer>, <article>, <p> etc.) δημιουργούμε την σελίδα μας, αλλά χωρίς την CSS η σελίδα μας όσο περιπλοκή σκέψη και να κρύβει από πίσω της και όσο δύσκολες τεχνολογίες και αν χρησιμοποιήθηκαν για την δημιουργίας της χωρίς την CSS δεν θα έχει νόημα διότι αυτή είναι που δίνει την ομορφιά σε μια σελίδα και προσθέτει ένα επιπλέον στοιχείο.

Η CSS έχει αλλάξει πολύ με τα χρόνια στην αρχή όταν δημιουργήθηκε μπορούσες να κάνεις μόνο απλά πράγματα όπως το να δίνεις διαφορετικά χρώματα και άλλα απλά πραγματάκια και με το πέρασμα των χρόνων και την ραγδαία ανάπτυξη της CSS κοινότητας αυτό άλλαξε και πλέον έχεις απεριόριστες δυνατότητες όπως το να δημιουργείς animation και να γράφεις κώδικα μέσα στην CSS μέσω της SASS.

6 RESPONSIVE WEB DESIGN

Όταν άρχισαν να φτιάχνονται οι πρώτες ιστοσελίδες το 90 ήταν πολύ απλές μόνο με κείμενο και λίγες εικόνες εκείνα τα χρόνια φαινόταν κάτι φοβερό και πρωτοπόρο αλλά τα χρόνια πέρασαν οι τεχνολογίες εξελίχθηκαν και οι απαιτήσεις μεγάλωσαν όπου φτάσαμε στις μέρες μας μια ιστοσελίδα να είναι πλέον μια ολόκληρη εταιρία.

Άρα μέσα στα χρόνια άλλαξε και ο τρόπος που οι προγραμματιστές σχεδιάζανε και υλοποιούσαν μια ιστοσελίδα πιο παλιά σχεδίαζαν τις ιστοσελίδες με τέτοιο τρόπο ώστε να λειτουργούν μόνο σε μεγάλες οθόνες(Desktop, Laptop) και δουλεύανε μόνο με px.

Κάτι όμως που άλλαξε συλλήβδην με την άνοδο των smartphone και tablet και οι ανάγκες τις αγοράς πλέον άλλαξαν διότι σύμφωνα με τελευταίες έρευνες ο κόσμος σερφάρει στο διαδίκτυο σε μεγαλύτερο ποσοστό από τα κινητά του κάτι που θα δούμε και στο παρακάτω διάγραμμα. Άρα και οι προγραμματιστές άλλαξαν εντελώς τον τρόπο σκέψης τους και προσανατολίστηκαν πρώτα στις μικρές συσκευές και μετά στις μεγαλύτερες.

Άρα η προσαρμογή της ιστοσελίδας ανάλογα με το μέγεθος τις οθόνης και το να αλλάζει το περιεχόμενο αυτόματα και να προσαρμόζεται ονομάζεται responsive design.

Το responsive design από όταν άρχισε γινόταν με συγκεκριμένος τρόπους, δηλαδή σχεδιάζει την ιστοσελίδα του και για μικρότερες συσκευές δημιουργεί media queries οπού ρυθμίζονται στην css και ανάλογα με την οθόνη ορίζουμε και διαφορετική css.

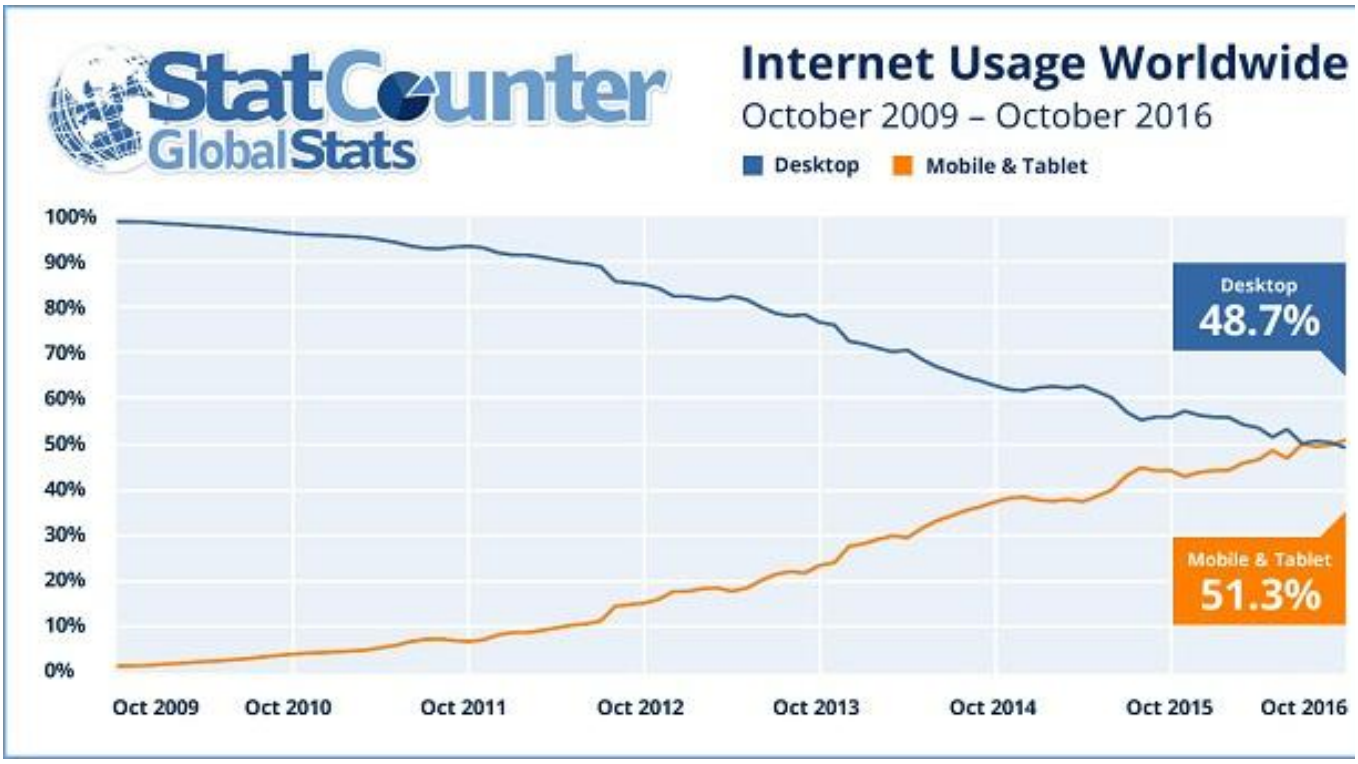
Αυτή η τεχνική όμως όπως μπορούμε να αντιληφθούμε μπορεί να είναι αποτελεσματική αλλά δημιουργεί κώδικα που επαναλαμβάνεται συνέχεια άρα τα τελευταία χρόνια οι developers άρχισαν να σχεδιάζουν τις ιστοσελίδες τους σε ems, rems, vw, vh όπου είναι κάποια στοιχεία τις css

που όμως αλλάζουν εντελώς τον τρόπο που φτιάχνουμε την σελίδα μας διότι είναι στοιχεία responsive και όχι προκαθορισμένες τιμές όπως τα px αλλά αλλάζουν τα μεγέθη τους ανάλογα με την οθόνη και να αποφεύγουμε τα πολλαπλά media queries.

Έτσι βλέπουμε ότι τα τελευταία χρόνια έχουν βγει πάρα πολλά css frameworks ώστε να μην χρειάζεται κάθε φορά να γράφουμε css αλλά με την προσθήκη κάποιων κλάσεων στην html να έχουμε αυτόματα το αποτέλεσμα και το site μας χωρίς πολύ δουλεία να είναι πλήρως responsive.

Αυτό φαίνεται να είναι πολύ εύκολο και απλό αλλά σε πολύ μεγάλα project είναι και πολύ επικίνδυνο να χρησιμοποιείται από αρχάριους διότι προσθέτουν κλάσεις χωρίς να ξέρουν ακριβώς που θα λειτουργήσουν στο DOM.

Από όλα αυτά τα framework δυο ξεχωρίζουν το Bootstrap που έχει φτιαχτεί από μηχανικούς του Twitter και το Foundation.



Εικόνα 1.4 Responsive VS Desktop.
(http://bgr.com/2016/11/02/internet-usage-desktop-vs-mobile/)

6.1 Διαφορές web applications με τα mobile applications

Τα web-application σε σχέση με τα mobile-application έχουν μεγάλες διαφορές και είναι δυο εντελώς διαφορετικά πράγματα στο κομμάτι του σχεδιασμού και τις υλοποιήσεις και χρησιμοποιούν εντελώς διαφορετικές αρχιτεκτονικές.

Αρχικά τα mobile-applications χρησιμοποιούν γλώσσες προγραμματισμού όπως Java, Objective-C, Swift και δουλεύουν σε είδη έτοιμες βιβλιοθήκες και ο κώδικας είναι αρκετά "βαρύς" και για αυτό χρησιμοποιούν συγκεκριμένα IDE σε σύγκρισή με τα web-application που γράφονται σε ένα απλό notepad, σε σχέση με τα web οπού γράφονται σε JavaScript, Ruby, Python.

Αλλά δεν έχουν διαφορές μόνο στο τρόπο της υλοποίησης αλλά και του σχεδιασμού διότι σχεδιάζονται εξολοκλήρου για μικρές συσκευές και όχι για Desktop διότι λειτουργούν μόνο σε συγκεκριμένα λειτουργικά συστήματα iOS, Android κάτι όμως που αλλάζει τα τελευταία χρόνια όπως βλέπουμε στα windows κινητά όπου οι εφαρμογές τους λειτουργούν και για τα laptop και desktop.

Επίσης στα mobile-application σε σύγκριση με τα web-application έχουν και εντελώς διαφορετική αίσθηση και στον τρόπο της επικοινωνίας με τον χρήστη και δίνουν μια πιο όμορφή και απαλή αίσθηση και με τα διάφορα animation χαρίζουν θεαματική εμπειρία στον χρήστη κάτι που προσπαθεί να αλλάξει από τους web-developers με όλα αυτά τα animation που έχουν ενσωματωθεί στην CSS3.

Όλο αυτό το σκηνικό όμως έχει αλλάξει εξολοκλήρου τα τελευταία χρόνια με την δημιουργία κάποιων libraries όπως το React Native όπου προσπαθούν να δώσουν στον χρήστη την δυνατότητα να προγραμματίζει με τα εργαλεία που προγραμματίζει στο web όπως την JavaScript,CSS,HTML πράγμα φοβερό και δίνοντας στις εφαρμογές ατελείωτες δυνατότητες όπως μέσα από το Progressive apps της Google να μπορεί να δημιουργεί για την web εφαρμογή του με δυνατότητες offline-mode.

Πιο αναλυτικά η React Native είναι ένα library που έχει δημιουργηθεί για να μπορούμε μα δημιουργούμε mobile εφαρμογές μέσω της JavaScript και μας προσφέρει αρκετά εργαλεία του web αλλά και του mobile(native).

7 ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ PROJECT

Πριν ξεκινήσουμε να βλέπουμε την εφαρμογή μας θα φτιάξουμε ένα νέο μικρό project και θα το ανεβάσουμε στο GitHub.

Άρα αρχικά κατεβάζουμε την Rails στον υπολογιστή μας και την κάνουμε εγκατάσταση από το ακόλουθο URL.

Ύστερα στον υπολογιστή μας αφού έχουμε εγκαταστήσει την rails μπορούμε να φτιάξουμε ένα νέο project γράφουμε rails new onomatouproject αφού εγκατασταθούν τα αρχεία μας μπαίνουμε στον φάκελο που μόλις δημιουργήσαμε cd onomatouproject και πατάμε rails server ώστε να ξεκινήσει ο development server και πατάμε στον browser μας localhost:3000 και θα πρέπει να βλέπουμε την παρακάτω εικόνα.

Εικόνα 1.5 Rails Hello World.

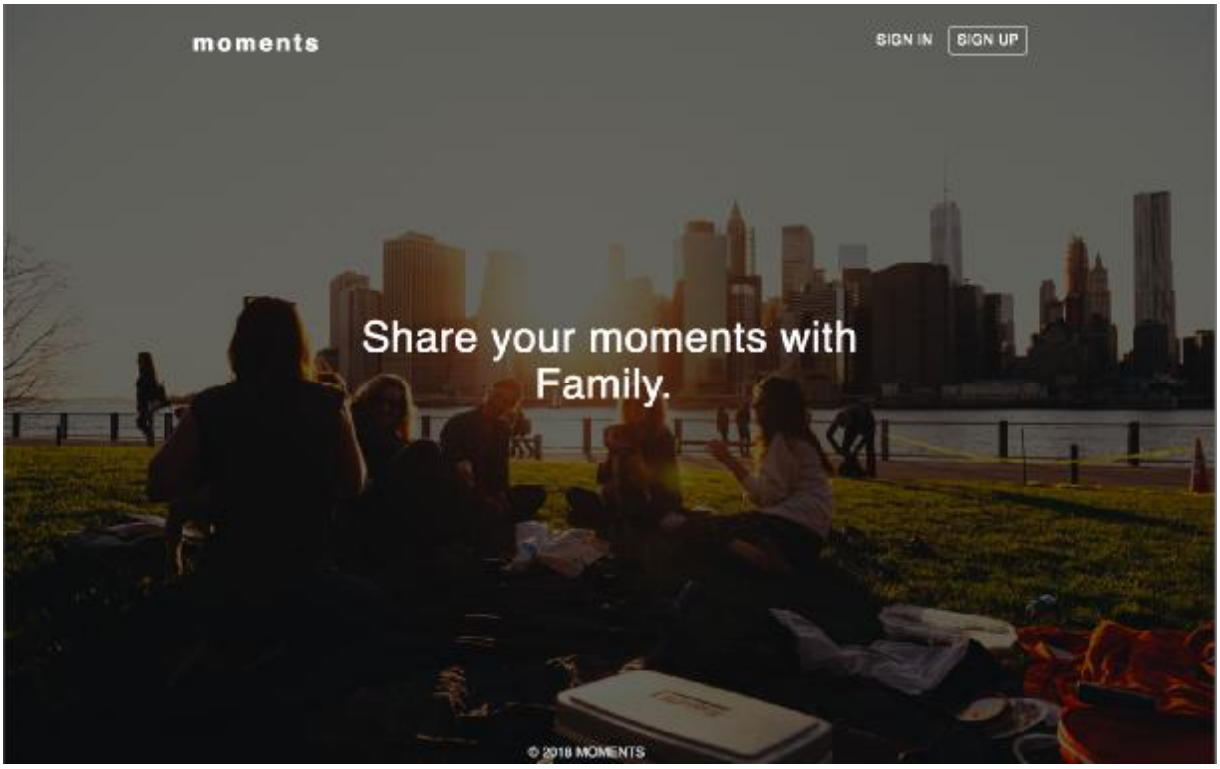
Μετά αφού κάνουμε τις πρώτες μας αλλαγές στο project ήρθε η ώρα να το ανεβάσουμε στο GitHub, το GitHub είναι ένα κοινωνικό δίκτυο για προγραμματιστές όπου ανεβάζουν project ανοιχτού.

Το πρώτο βήμα είναι να κατεβάσουμε το git εργαλείο από αυτό το site https://git-scm.com/ αφού το έχουμε εγκαταστήσει πρέπει να ακολουθήσουμε κάποιες εντολές ώστε το project μας να ανέβει στο GitHub.

Πρώτα φτιάχνουμε ένα νέο repository από το site του GitHub ύστερα αφού έχουμε φτιάξει το project π.χ. με όνομα foo Bar είμαστε έτοιμοι να το ανεβάσουμε

```
1) cd /path/to/your/repo
2) git remote add origin https://geobde@github.com/geobde/foobar.git
3) git commit -am “Πρώτο ανέβασμα στο GitHub”
4) git push -u origin master
```

8 Η ΕΦΑΡΜΟΓΗ MOMENTS



Εικόνα 1.6 Moments Αρχική Σελίδα

Η εφαρμογή moments είναι μια web εφαρμογή κοινωνικών δικτύων όπου μπορούν οι χρήστες να εγγραφούν στην πλατφόρμα να ανεβάσουν φωτογραφίες και να ακολουθήσουν άλλους χρήστες. Η εφαρμογή δουλεύει και στα παραδοσιακά μέσα(Desktop, Laptop) αλλά και στα smartphone.

Αρχικά σχεδιάσαμε την εφαρμογή μας σε διαφορετικά στάδια στο χαρτί ώστε να θέσουμε κάποια πλαίσια, μετά φτιάξαμε τις οντότητές μας και τις σχέσεις που θα έχουν μεταξύ τους.

Ύστερα έπρεπε να επιλέξουμε σε ποια τεχνολογία θα υλοποιήσουμε την εφαρμογή ανάμεσα σε τόσα πολλά frameworks και libraries και καταλήξαμε στην Ruby on Rails λόγω των δημοφιλών εφαρμογών που είχαν φτιαχτεί στο συγκεκριμένο framework αλλά και λόγω του μεγάλου community διότι σε ένα μεγάλο project σίγουρα θα συναντούσαμε και δυσκολίες.

Μετά ξεκινήσαμε την υλοποίηση του project όπου έπρεπε αρχικά να μάθουμε σε βάθος την Ruby για να κατανοήσουμε διαφορές και ομοιότητες από τις γλώσσες που ήδη ξέραμε δηλαδή την JavaScript και C, αλλά και στην διαφορετική αρχιτεκτονική που χρησιμοποιεί η Rails στο κομμάτι της βάσης χρησιμοποιώντας την Active Record.

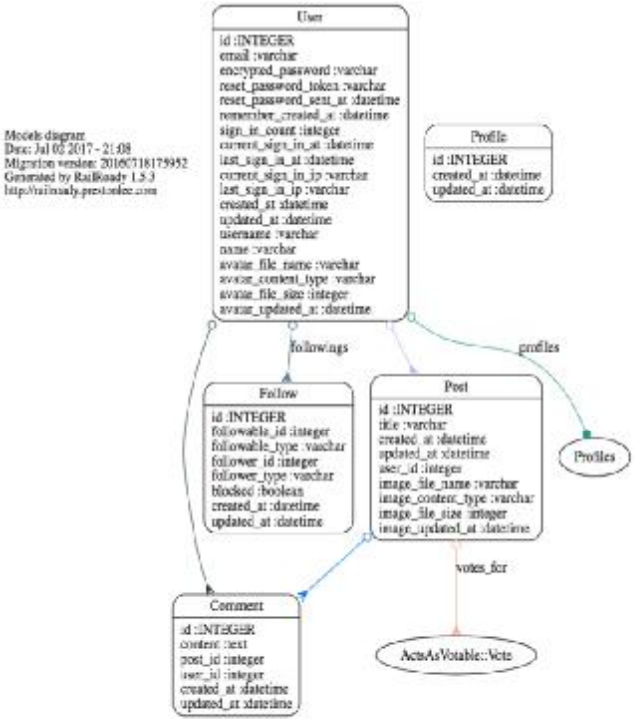
Άρα μαθαίνοντας αρκετά πράγματα για το οικοσύστημα της Ruby on Rails αρχίσαμε την τελική υλοποίηση, η εφαρμογή όπως και όλες η Rails εφαρμογές χρησιμοποιούν την αρχιτεκτονική MVC, άρα αρχίσαμε να φτιάχνω τα μοντέλα μου(κλάσεις, οντότητες).

Η εφαρμογή αποτελείται από τους χρήστες(Users) όπου είναι το βασικό κομμάτι, τα Posts οπού είναι οι δημοσιεύσεις των χρηστών, τα σχόλια(Comments) των χρηστών σε κάθε δημοσίευση. Επίσης αποτελείται από την οντότητα Follow/Unfollow οπού οι χρήστες θα ακολουθούνται μεταξύ τους ώστε στον τοίχο τους(Αρχική) να εμφανίζονται οι δημοσιεύσεις μόνο των ατόμων που ακολουθούν.

Ο κώδικας που αναφέρεται πιο κάτω θα χρησιμοποιεί το μοντέλο MVC και θα σχολιάζεται ενδελεχώς.

8.1 Η δομή της εφαρμογής

Εικόνα 1.7 Moments Δομή



Στο παραπάνω διάγραμμα βλέπουμε όλες της οντότητες της βάσης μας το πώς συνδέονται αλλά και της και απο ποιά στοιχεία αποτελούνται.

Πιο αναλυτικά βλέπουμε ο κορμός της εφαρμογής μας είναι οι χρήστες(Users) και συνδέονται με τα αρθρα(Posts) και έχουν μια σχέση ενα προς πολλά δηλαδή ενας χρήστης μπορεί να δημιουργήσει πολλά άρθρα.
Ύστερα συνδέεται και με τα σχόλια(Comments) όπου έχει την ίδια σχέση
αλλά την ίδια σχέση με τα comments εχουν και τα posts δηλαδή ένα άρθρο μπορεί να έχει πολλά comments,τα posts ομως συνδέονται και με της ψήφους(Vote).

Τέλος βλέπουμε οι χρήστες να συνδέονται και με τα προφιλ(Profiles) και να έχουν μια σχέση ένα πρός ενα διότι ένας χρήστης μπορεί να έχει μόνο ενα προφίλ.

8.2 Οι Χρήστες

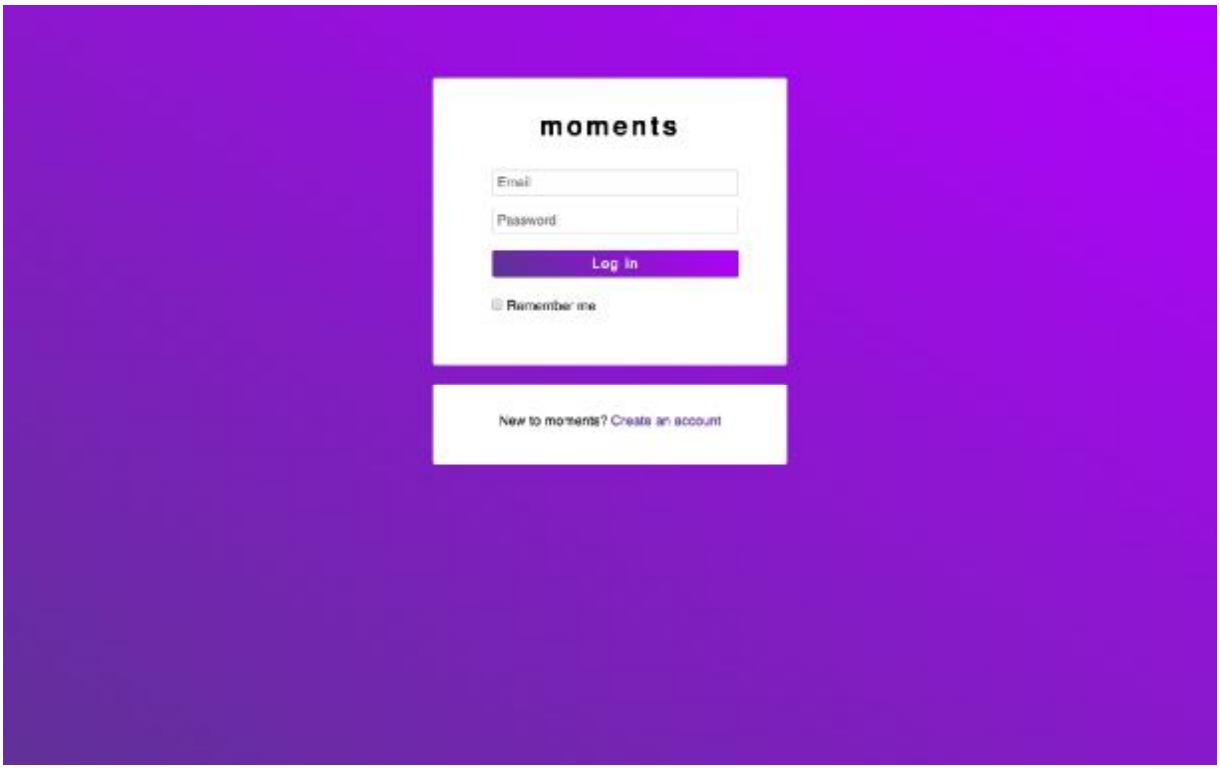
Οι χρήστες είναι η βασική μας οντότητα στην εφαρμογή μας διότι όλα περιστρέφονται γύρω από αυτούς όπως και σε κάθε εφαρμογή κοινωνικού δικτύου.

Στο moments οι χρήστες μπορούν να ανεβάζουν φωτογραφίες να ακολουθούν μόνο άτομα με τα οποία θέλουν να βρίσκονται σε επικοινωνία να σχολιάζουν και να κάνουν like σε posts άλλων χρηστών.

Για την πιο σωστή λειτουργία χρησιμοποιούμε το devise gem το οποίο αυτοματοποιεί την λειτουργία των χρηστών.

Για να αποκτήσεις πρόσβαση στην εφαρμογή πρέπει πρώτα να κάνεις μια εγγραφή και να συμπληρώσεις τα στοιχεία σου.

8.2.1 Είσοδος των χρηστών στην εφαρμογή



Εικόνα 1.8 Moments Είσοδος στην εφαρμογή

Εκτός από το validation στην βάση που γίνεται μέσω της Active Record
όπου θέτουμε όρια σε σχέση με το τι θα δέχεται η εφαρμογή από τον χρήστη άλλα μέσα στην κλάση μας κάνουμε και τις απαραίτητες συσχετίσεις με τις υπόλοιπές κλάσεις όπως τα post,comments,profiles etc.

Επιπλέον θέτουμε όρια και στο τι μέγεθος και τύπο εικόνα θα ανεβάσει ο χρήστης.

```
(app/models/user.rb)
class User < ActiveRecord::Base
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable
  has_many :posts
  has_many :comments
  has_one :profiles
  acts_as_follower
  acts_as_followable
  validates :name, presence: true, length: {maximum: 50}
  validates :username, presence: true, length: {maximum: 30}
  has_attached_file :avatar, styles: { medium: "100%>", small: "100%>" }
  validates_attachment_content_type :avatar, content_type: /\Aimage\/.*\Z/
```

Αλλά επειδή πρέπει να δίνουμε στον χρήστη και μια αλληλεπίδραση και να καταλαβαίνει δυναμικά το λάθος που μπορεί να κάνει κατά την εγγραφή του

έχω φταίξει και τα validation στην JavaScript όπου ενεργοποιούνται κατά την εγγραφή του χρήστη:

(app/assets/javascripts/validation.js)

```
var simpleName = document.getElementById('name');
var userName = document.getElementById('username');
var userEmail = document.getElementById('email');
var userPass = document.getElementById('password');
var userPassConfirm = document.getElementById('password-confirm');
var valMessageSimpleName = document.getElementById('message-name');
var valMessageName = document.getElementById('message-username');
var valMessageEmail = document.getElementById('message-email');
var valMessagePass = document.getElementById('message-pass');
var valMessagePassConfirm = document.getElementById('message-pass-confirm');
var submitButton = document.getElementById('submit-form');

/**
* Function to validate username if user puts a name
* smaller than 6 characters show error message otherwise
* show succeed message.
*/
function validationName() {
  var name = simpleName.value;
  if(name.length < 6) {
    valMessageSimpleName.className = "failed";
    valMessageSimpleName.textContent = "Mininum 6 characters name!";

  }
  else {
    valMessageSimpleName.className = "succeed";
    valMessageSimpleName.textContent = "Your name is valid.";

  }
}


/**
* Function to validate username if user puts a name
* smaller than 6 characters show error message otherwise
* show succeed message.
*/
function validationUsername() {
  var name = userName.value;
  if(name.length < 3) {
    valMessageName.textContent = "Mininum 3 characters username!";
    valMessageName.className = "failed";
  }
  else {
      valMessageName.textContent = "Your username is valid.";
      valMessageName.className = "succeed";
  }
```

```
}

/**
* Function to validate email if user left email
* blank  show error message otherwise show succeed
* message.
*/
function validationEmail() {
  var email = userEmail.value;
  if(email.length == 0) {
    valMessageEmail.textContent = "Can't leave email blank!";
    valMessageEmail.className = "failed";
  }else {
       valMessageEmail.textContent = "Your email is valid.";
       valMessageEmail.className = "succeed";
  }
}

/**
* Function to validate password if user puts a password
* smaller than 8 characters show eror message otherwise
* show succeed message.
*/
function validationPassword() {
  var pass= userPass.value;

  if(pass.length < 8) {
    valMessagePass.textContent = "Mininum 8 characters passwords!";
    valMessagePass.className = "failed";
  }
  else {
       valMessagePass.textContent = "Your password is valid.";
       valMessagePass.className = "succeed";
  }
}

/**
* Function to validate if the password confirmation
* is the same with password.
*/
function validationPassConfirm() {
       var pass= userPass.value;
  var confirm = userPassConfirm.value;

  if(confirm != pass || confirm.length == 0) {
       valMessagePassConfirm.textContent = "Password don't match!";
       valMessagePassConfirm.className = "failed";
  }else {
       valMessagePassConfirm.textContent = "Your password is match.";
       valMessagePassConfirm.className = "succeed";
  }
}

/**
* Function to validate if user leave any field
```

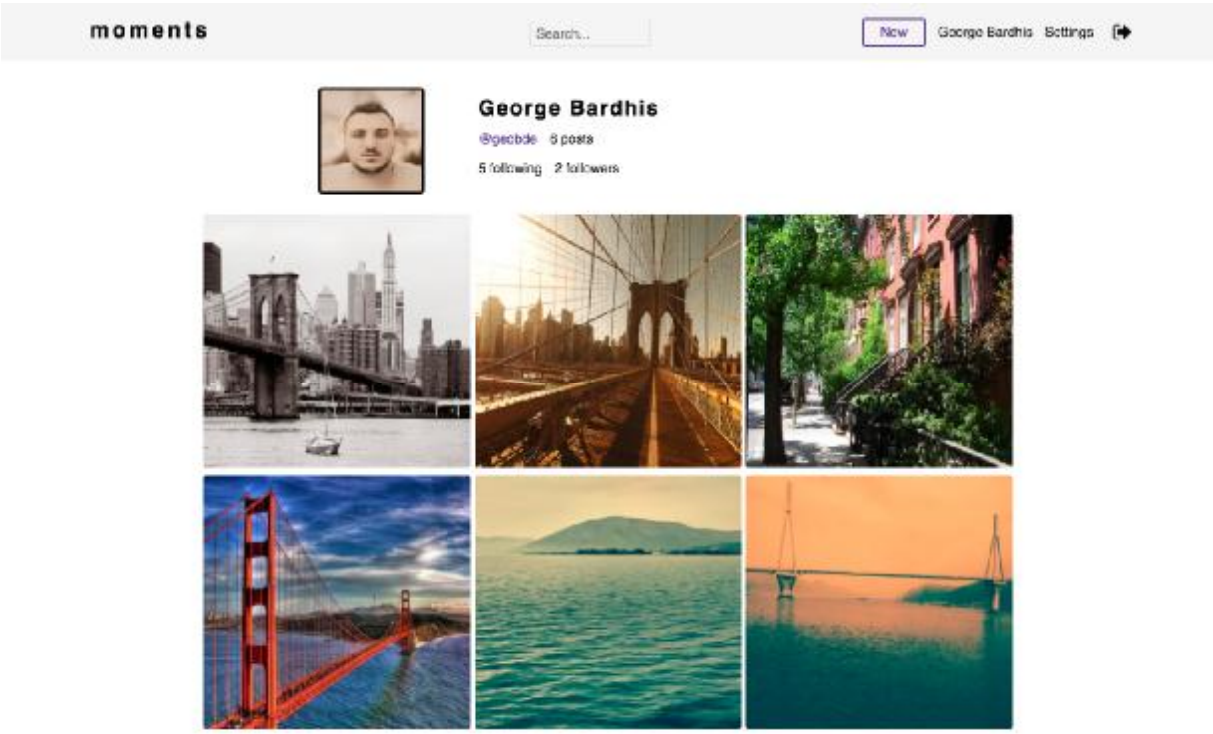
```
* blank show eror message otherwise show succeed
* message.
*/
function validationForm() {
  var name = userName.value;
  var email = userEmail.value;
  var pass = userPass.value;
  var confirm = userPassConfirm.value;

  if(name.length == 0 || email.length == 0 || pass.length == 0 || confirm.length ==
0) {
        alert("You can't leave  blank field!");
  }else {
 }
}

/**
* Puts events for username, email, password and
* password confirmation and in submit button.
*/
simpleName.addEventListener('blur', validationName);
userName.addEventListener('blur', validationUsername);
userEmail.addEventListener('blur', validationEmail);
userPass.addEventListener('blur', validationPassword);
userPassConfirm.addEventListener('blur', validationPassConfirm);
submitButton.addEventListener('click', validationForm);
```

Ακριβώς η ίδια λογική ακολουθείτε και κατά την είσοδο τον χρηστών στην εφαρμογή μετά την εγγραφή τους συμπληρώνοντας το email εγγραφής και τον κωδικό τους.

8.2.2 Προφίλ των χρηστών



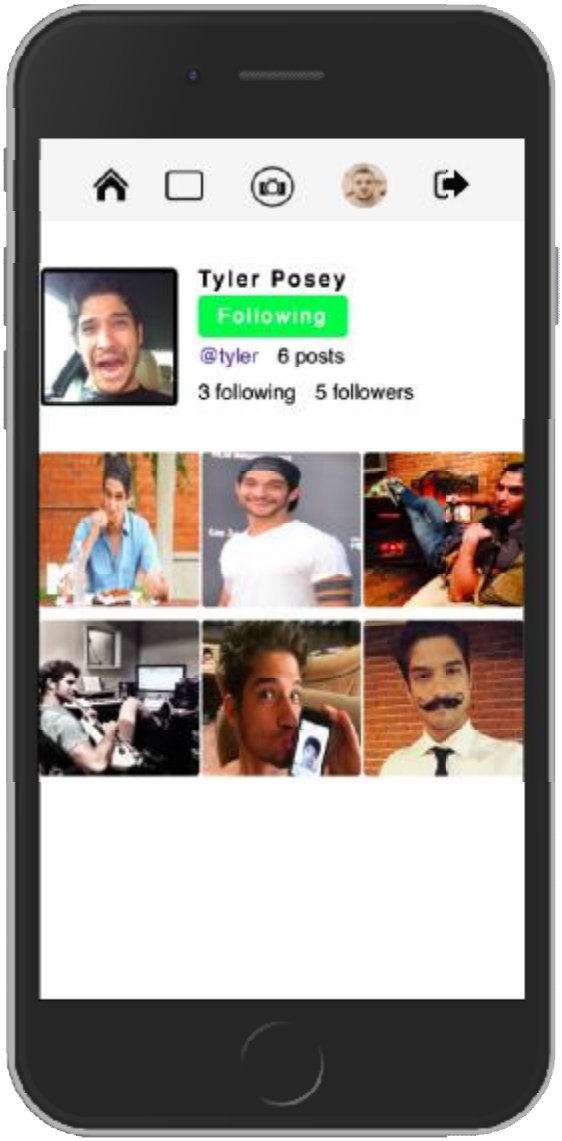
Εικόνα 1.9 Προφίλ των χρηστών

Αυτή είναι η προβολή του προσωπικού προφίλ του χρήστη όπου του εμφανίζει την φωτογραφία του, το όνομα του, το username
του και τα άτομα του ακολουθεί αλλά και από πόσα άτομα ακολουθείτε.

Επίσης εμφανίζει και τα post του χρήστη δηλαδή τις φωτογραφίες που έχει ανεβάσει στο moments.

Στην πάνω εικόνα βλέπουμε το πώς εμφανίζεται το προφίλ ενός άλλου χρήστη που ακολουθούμε οι διαφορές είναι ότι τώρα βλέπουμε ένα πλαίσιο όπου αναφέρει το Following δηλαδή ότι ακολουθώ αυτόν τον χρήστη και μπορώ να σχολιάζω και κάνω like στα post που ανεβάζει.

8.3 Responsive Profile Page



Εικόνα 2.0 Responsive Προφίλ των χρηστών

Responsive προβολή από iPhone 6 του προφίλ στις πάνω εικόνες βλέπουμε τις μεγάλες διαφορές τις σελίδας μου πλέον έχει αλλάξει ολοκληρωτικά η δομή και έχει μετατραπεί σε μία εφαρμογή για κινητά
αυτό υλοποιείται μέσω της CSS και των media queries όπου θα το δούμε αναλυτικά στον κώδικα πιο κάτω.

Άρα για την προβολή των στοιχείων του κάθε χρήστη στην profile page είναι υπεύθυνη η πιο κάτω function η οποία αρχικά αναγνωρίζει τον χρήστη από το id του μετά εμφανίζει τα post μόνο του συγκεκριμένου χρήστη.

```
(app/controllers/profiles_controller.rb)

  def show
    @user = User.find(params[:id])
    @posts = @user.posts.paginate(page: params[:page], per_page:
20).order('created_at DESC')
    if params[:search]
     @users = User.search(params[:search])
   end
```

Για την λειτουργία follow/unfollow μεταξύ των χρηστών στην εφαρμογή
είναι υπευθυνες οι πιο κάτω functions και χρησιμοποιώ το gem acts_as_follower το οποίο μέσα από τους controllers του λεμέ πως πρέπει να λειτουργεί και να κάνει τις συσχετίσεις.

```
(app/controllers/profiles_controller.rb)

def follow
      @user = User.find(params[:id])
      current_user.follow(@user)
      redirect_to :back
  end

  def unfollow
      @user = User.find(params[:id])
      current_user.stop_following(@user)
      redirect_to :back
  end
```

Η οντότητα(Model) της λειτουργίας follow/unfollow κάνει όλες τις κατάλληλες συνδέσεις ώστε να μην δημιουργείτε κάποιο conflict μεταξύ των χρηστών.

```
(app/models/profile.rb)

class Follow < ActiveRecord::Base

  extend ActsAsFollower::FollowerLib
  extend ActsAsFollower::FollowScopes

  # NOTE: Follows belong to the "followable" interface, and also to followers
  belongs_to :followable, :polymorphic => true
  belongs_to :follower,   :polymorphic => true

  def block!
    self.update_attribute(:blocked, true)
  end

end
```

Η προβολή της συγκεκριμένης σελίδας δηλαδή το View apo to MVC
είναι ο κώδικας πιο κάτω:

(app/views/profiles/show.html.erb)

```
 <html>
<head>
  <title><% provide(:title, @user.name) %></title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
 <div class="header-con">
   <%= link_to "moments", root_path, class: 'logo-con'%>
   <div class="search-bar">
   <%= form_tag(profiles_path, :method => "get") do %>
      <%= text_field_tag :search, params[:search], placeholder: "Search...", class:
'search' %>
    <% end %>
    </div>
     <ul class="navbar-con">
        <li><%= link_to "New", new_post_path, class: 'new-post' %></li>
        <li><%= link_to current_user.name, profile_path(current_user)%></li>
        <li><%= link_to "Settings", edit_user_registration_path %></li>
        <li class="sign-out-container"><%= link_to(image_tag('sign_out.png', class:
'sign-out'),destroy_user_session_path, method: :delete) %></li>
     </ul>
     <div class="responsive-navbar">
       <div class="small-device-home"><%= link_to(image_tag('house.png', class:
'responsive-home'), root_path)%></div>
       <div class="responsive-search">
       <%= form_tag(profiles_path, :method => "get") do %>
         <%= text_field_tag :search, params[:search], class: 'res-search' %>
       <% end %>
       </div>
       <div class="small-device-new"><%= link_to(image_tag('new_image.png', class:
'responsive-new'), new_post_path)%></div>
       <div class="small-device-img"><%=
link_to(image_tag(current_user.avatar(:small), class: 'responsive-img'),
profile_path(current_user))%></div>
       <div class="small-device-sign-out"><%= link_to(image_tag('sign_out.png',
class: 'responsive-sign-out'),destroy_user_session_path, method: :delete) %></div>
      </div>
 </div>
<div class="profile-container">
  <div class="profile-wrapper">
   <div class="flex-info">
    <div class="avatar-container">
       <%= image_tag @user.avatar.url(:medium), class: 'avatar' %><br><br>
     </div>
     <div class="info-container">
       <div class="follow-name-flex">
        <div class="profile-name"><%= @user.name%></div>
        <div class="follow-container">
```

```
        <% if current_user != @user %>
          <% if current_user.following?(@user) %>
                <%= link_to("Following", unfollow_profile_path(@user), class:
'following') %>
            <% else %>
                <%= link_to("Follow" , follow_profile_path(@user), class: 'follow')%>
            <% end %>
           <% end %>
        </div>
        </div>
       <div class="follow-name-info">
        <div class="profile-username">@<%= @user.username %></div>
        <div class="posts-count"><%= @user.posts.count %> posts</div>
        <% if current_user == @user %>
        <div class="dots"><%= link_to("...", edit_user_registration_path,  class:
'dots-style') %></div>
        <% end %>
       </div>
       <div class="followers-following">
        <div class="following-users"><%= @user.all_follows.length %> following</div>
        <div class="followers-users"><%= @user.followers.length %> followers</div>
        </div>
      </div>
    </div>
        <div class="posts-container">
         <% @posts.each do |post| %>
          <div class="post-container">
          <%= link_to(image_tag(post.image.url(:medium), class: 'post-image'),
post_path(post)) %>
          </div>
         <% end %>
         </div>
         <div class="paginate">
           <%= will_paginate @posts,  :page_links => false,  previous_label:"« Prev
", next_label:" Next »"%>
        </div>
        </div>
      </div>
  </body>
</html>
```

Όπου το πιο σημαντικό κομμάτι είναι η προβολή διαφορετικού status(Follow/Following) ανάλογα με το εάν έχουμε κάνει η όχι τον συγκεκριμένο χρήστη

```
<% if current_user != @user %>
       <% if current_user.following?(@user) %>
           <%= link_to("Following", unfollow_profile_path(@user), class:
'following') %>
  <% else %>
           <%= link_to("Follow" , follow_profile_path(@user), class: 'follow')%>
         <% end %>
   <% end %>
```

Ένα άλλο σημαντικό κομμάτι είναι και το pagination δηλαδή τα πόσα post θα εμφανίζονται στην σελίδα, διότι αν το αφήναμε ελεύθερο θα αντιμετωπίζαμε μεγάλο πρόβλημα performance και λειτουργικότητας.

Αυτό επιτυγχάνεται μέσα από το gem will_paginate και το βλέπουμε στον κώδικα πιο κάτω:

```
   <div class="paginate">
           <%= will_paginate @posts,  :page_links => false,  previous_label:"« Prev
", next_label:" Next »"%>
```

Κώδικας για την προβολή δηλαδή το View από το MVC για την τροποποίηση των στοιχείων:

```
(app/views/devise/edit.html.erb)

 <div class="form-container-edit">
    <h2 class="form-title">Settings</h2>
    <%= form_for(resource, as: resource_name, url: registration_path(resource_name),
html: { method: :put }) do |f| %>

  <div class="content-container-sign">
   <div class="field-group">
     <%= f.text_field :name, placeholder: 'Name', class: 'input-text', id: 'name' %>
     <div id="message-name"></div>
   </div>

  <div class="field-group">
    <%= f.text_field :username, placeholder: 'Username', class: 'input-text', id:
'username'%>
    <div id="message-username"></div>
  </div>

  <div class="field-group">
    <%= f.email_field :email,  placeholder: 'Email', class: 'input-text', id: 'email'
%>
    <div id="message-email"></div>
  </div>

 <% if devise_mapping.confirmable? && resource.pending_reconfirmation? %>
    <div>Currently waiting confirmation for: <%= resource.unconfirmed_email %></div>
  <% end %>

  <div class="field-group">
    <%= f.password_field :password,  placeholder: 'Password', class: 'input-text',
id: 'password' %>
    <div id="message-pass"></div>
  </div>

  <div class="field-group">
```

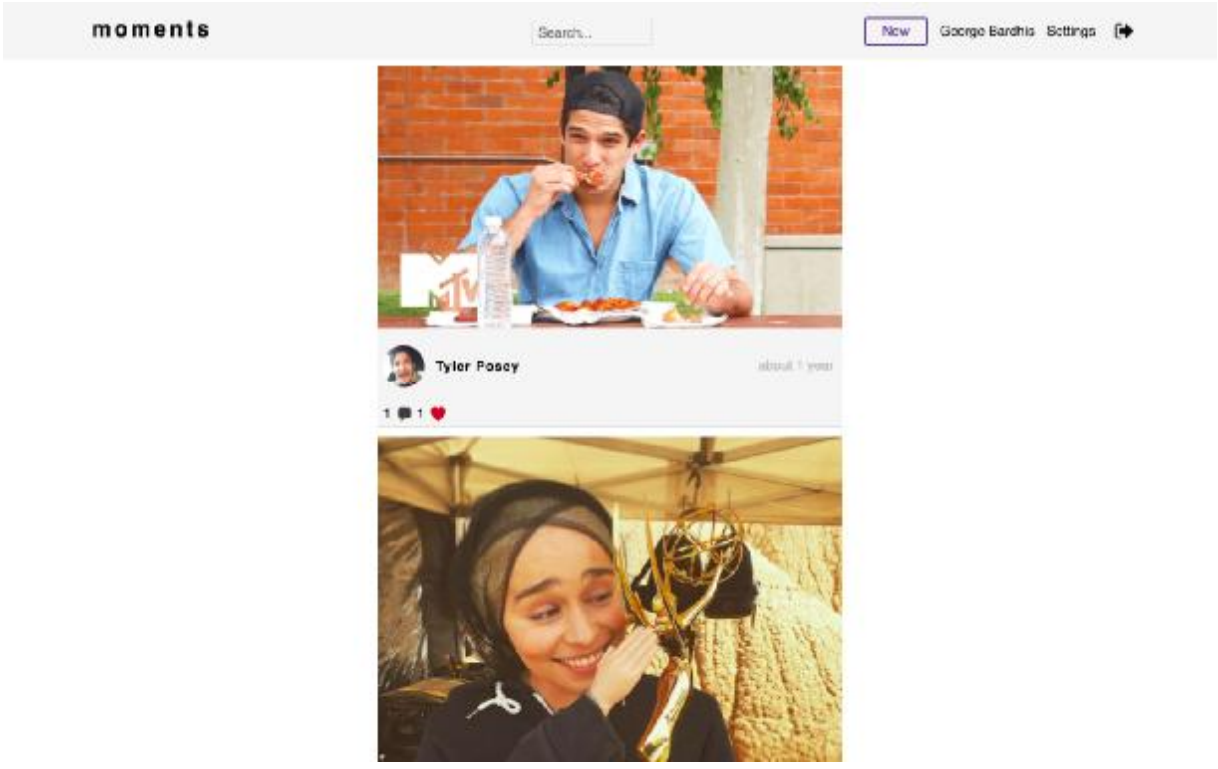
```
    <%= f.password_field :password_confirmation, placeholder: 'Password
Confirmation', class: 'input-text', id: 'password-confirm' %>
    <div id="message-pass-confirm"></div>
  </div>

<div class="field-group">
  <%= f.password_field :current_password, placeholder: 'Current Password', class:
'input-text'%>
</div>

<div class="field-group">
   <%= f.file_field :avatar, class: 'input-file'%>
  </div>

  <div class="actions-group">
    <%= f.submit "Update",  class: 'input-submit', id: 'submit-form' %>
  </div>
  <% end %>
</div>
```

8.4 Οι Δημοσιεύσεις



Εικόνα 2.1 Δημοσιεύσεις των χρηστών
Οι δημοσιεύσεις / posts είναι ένα μεγάλο κομμάτι της εφαρμογής διότι οι χρήστες αλληλοεπιδρούν δυναμικά και ανταλλάσσουν φωτογραφίες και σχόλια.

Επίσης στο κομμάτι της υλοποίησης και του κώδικα ήταν και η οντότητα που συνδεόταν με όλες τις υπόλοιπες και ήταν ο βασικός κορμός των Models πιο κάτω θα δούμε την δομή και υλοποίηση στο κομμάτι του κώδικα.

Ξεκινάμε με το M (models) απο το MVC.

```
(app/models/posts.rb)

class Post < ActiveRecord::Base
       belongs_to :user
       belongs_to :profiles
       has_many :comments
       acts_as_votable
       scope :subscribed, ->( all_following) { where user_id: all_following }
       has_attached_file :image, styles: { medium: "100%>", small: "100%>" }
  validates_attachment_content_type :image, content_type: /\Aimage\/.*\Z/
end
```

Άρα βλέπουμε την κλάση μας όπου συνδέεται με όλες τις υπόλοιπες και ορίζουμε το είδος των σχέσεων που έχουν μεταξύ τους.

Αλλά πολύ σημαντικό κομμάτι στον τρόπο που θα λειτουργούν οι δημοσιεύσεις παίζουν οι controllers που είναι και η λογική της κάθε λειτουργίας.

Άρα θα δούμε το C (controllers) από το MVC.

Ξεκινάμε με την κλάση μας Posts οπού μέσα σε αυτή θα γράψουμε όλες τις λειτουργίες τής κλάσης και το πώς θα πρέπει να συμπεριφέρεται.

```
(app/controllers/posts_controller.rb)

class PostsController < ApplicationController
       before_action :find_post, only:[:show, :edit, :update, :upvote, :destroy]

       before_action :authenticate_user!
```

Άρα η πρώτη function είναι υπεύθυνη για την αρχική μας σελίδα όπου είναι όλα τα posts

```
def index
 @posts = Post.paginate(page: params[:page], per_page: 20).order('created_at
DESC').subscribed current_user.all_following
     @users = User.all
     @user = current_user
          if params[:search]
```

```
                @users = User.search(params[:search])
            end
 end
```

Μετά έχουμε την show η οποία ευθύνεται για την προβολή ενός post από κάποιον χρήστη

```
       def show
            @comments = Comment.where(post_id: @post)
            @users = User.all
            @user = current_user
    end
```

Ύστερα έχουμε την new οπου ευθύνεται για την δημιουργία ενος άρθρου

```
 def new
            @post = current_user.posts.build
            @users = User.all
            @user = current_user
end
```

Μέτα έχουμε την create που ουσιαστικά είναι συνδεδεμένη με την show και δημιουργεί το post μας

```
      def create
            @post = current_user.posts.build(post_params)

            if @post.save
                  redirect_to @post
            else
             render 'new'
            end
      end
```

Πιο κάτω έχουμε την edit και update που ευθύνονται για την τροποποίηση κάποιου post

```
  def edit
    if params[:search]
      @users = User.search(params[:search])
    end
 end
```

```
   def update
            if @post.update(post_params)
                  redirect_to @post
```

```
            else
              render 'edit'
            end
       end
```

'Υστερα έχουμε την destroy για την διαγραφή του post

```
    def destroy
         if current_user == @post.user
              @post.destroy
              redirect_to root_path
             else
              redirect_to @post
           end
      end
```

Επίσης έχουμε άλλη μια function που θα την δούμε πιο κάτω σε άλλο κομμάτι που ευθύνεται για τα like των posts

```
     def upvote
        @post.upvote_by current_user
        redirect_to @post
    end
```

Τέλος θα δούμε και δυο private functions όπου τις έχουμε δημιουργήσει για να μην επαναλαμβάνουμε συνέχεια τον ίδιο κώδικα(DRY)

```
    def find_post
            @post = Post.find(params[:id])
       end

       def post_params
            params.require(:post).permit(:title, :image)
   end
```

Άρα πάμε στο τρίτο κομμάτι του MVC το V (view) και θα δούμε την προβολή των σελίδων μας όπου είναι συνδεδεμένες οι λειτουργίες τους με τον controller και την αντίστοιχη function

Πιο κάτω ακολουθεί ο κώδικας για την αρχική σελίδα των δημοσιεύσεων.

(app/views/posts/index.html.erb)

```
<div class="main-container">
   <div class="main-wrapper">
    <% @posts.each do |post| %>
     <div class="moment-container">
     <div class="moment-image-container"><%= image_tag post.image.url(:medium),
class: 'moment-image' %></div>
        <div class="moment-flex-one">
          <div class="moment-avatar-container"><%= image_tag
post.user.avatar(:small), class: 'moment-avatar' %></div>
          <%= link_to post.user.name,  profile_path(post.user), class: 'moment-
name'%>
          <div class="time-created"><%= time_ago_in_words(post.created_at) %></div>
        </div>
        <div class="flex-two">
          <div class="comments-count"> <%= post.comments.count %></div>
          <%= image_tag 'comments.png', class: 'comments-image' %>
          <div class="like-count"><%= post.get_upvotes.size %></div>
          <div class="like-image-comment">
          <%= link_to(image_tag('like.png', class: 'like-image'),
like_post_path(post), method: :put)%>
          </div>
        </div>
      </div>
      <% end %>
      <div class="paginate">
        <%= will_paginate @posts,  :page_links => false,  previous_label:"« Prev ",
next_label:" Next »"%>
     </div>
     </div>
   </div>
```

Στην αρχική σελίδα μας δίνεται και η δυνατότητα να κάνουμε σε όποιο post θέλουμε like μέσω του παρακάτω κώδικα

```
   <div class="like-image-comment">
          <%= link_to(image_tag('like.png', class: 'like-image'),
like_post_path(post), method: :put)%>
 </div>
```

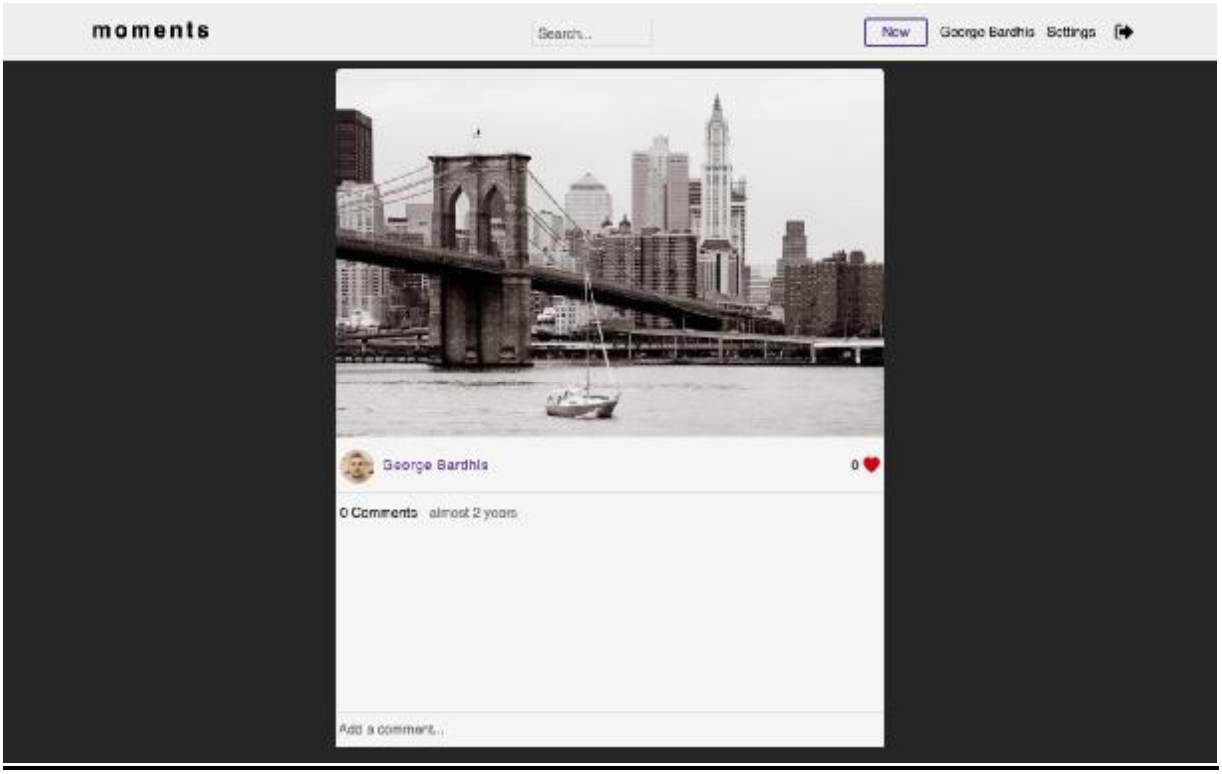
Όπου συνδέεται με την function που είχαμε δει πιο πάνω στο posts controller

```
 def upvote
```

```
        @post.upvote_by current_user
        redirect_to @post
end
```

Όπως βλέπουμε μέσω της εμφωλευμένης ruby στην html προβάλουμε όλα τα post όλων των χρηστών που ακολουθούμε (Follow) οπού το κάθε post ξεχωριστά αποτελείτε από την κεντρική φωτογραφία του post, το όνομα του χρήστη και την προσωπική φωτογραφία, την χρονική στιγμή που το ανέβασε και τον αριθμό των comments και like που έχει το συγκεκριμένο post.

8.4.1 Δημοσίευση του χρήστη



Εικόνα 2.2 Δημοσίευση του χρήστη

Η παραπάνω εικόνα είναι η προβολή του post που ανεβάζει ο χρήστης όπως βλέπουμε έχει μια κεντρική εικόνα δηλαδή το post του χρήστη και κάτω εμφανίζεται το όνομα το χρήστη τα likes που έχει πάρει η φωτογραφία, τα comments και ο χρόνος που ανέβηκε.

Στον παρακάτω κώδικα είναι η προβολή της σελίδας

```
(app/views/posts/show.html.erb)

<div class="full-post-container">
    <div class="one-post-container">
     <div class="full-image-container">
      <%= image_tag @post.image.url(:medium), class: 'full-image' %><br>
     </div>
     <div class="full-comments-container">
     <div class="head-post-container">
          <%= image_tag @post.user.avatar(:small), class: 'moment-avatar-comment' %>
          <div class="head-post-name"><%= link_to @post.user.name,
profile_path(@post.user), class: 'post-name'%></div>
           <div class="like-count-cont">
             <div class="like-count-comments"><%= @post.get_upvotes.size %></div>
             <%= link_to(image_tag('like.png', class: 'like-image-comment'),
like_post_path(@post), method: :put)%>
           </div>
     </div>
     <div class="border"></div>

      <div class="post-title"><%= @post.title %></div>
    <div class="comments-time-created">
      <%= pluralize(@post.comments.count, "Comment") %>
      <div class="time-created-full-post"><%= time_ago_in_words(@post.created_at)
%></div>
    </div>
     <div class="all-comments">
      <% @comments.each do |comment| %>
      <div class="main-post-container">
       <%= link_to comment.user.username, profile_path(comment.user), class: 'post-
username'%>
      <div class="post-content"><%= comment.content %></div>
      </div>
      <% end %>
      </div>
       <%= render 'comments/comment_form' %>
      </div>
     </div>
```

8.4.2 Σχολιασμός των χρηστών

Το νέο στοιχείο πάνω στον κώδικα είναι τα comment διότι είναι η μόνη από τις σελίδες όπου μπορούν οι χρήστες να σχολιάζουν την φωτογραφία αυτό επιτυγχάνεται μέσω του παρακάτω κώδικα και ουσιαστικά συνδέεται με την function show στους controllers του post όπως είδαμε και παραπάνω.

```
div class="comments-time-created">
      <%= pluralize(@post.comments.count, "Comment") %>
```

```
      <div class="time-created-full-post"><%= time_ago_in_words(@post.created_at)
%></div>
    </div>
     <div class="all-comments">
      <% @comments.each do |comment| %>
      <div class="main-post-container">
       <%= link_to comment.user.username, profile_path(comment.user), class: 'post-
username'%>
      <div class="post-content"><%= comment.content %></div>
      </div>
      <% end %>
      </div>
       <%= render 'comments/comment_form' %>
 </div>
```

Άρα προβάλουμε μέσω της ruby και μέσω της each μεθόδου όλα τα σχόλια που έχει συγκεντρώση η εικόνα και σε άλλη σελίδα έχω την φόρμα μου όπου την κάνω render σε αυτήν την σελίδα για λόγους σωστού κώδικα άρα ο κώδικας της φόρμας για την προσθήκη νέου σχολίου είναι παρακάτω.

```
<%= form_for([@post, @post.comments.build]) do |f| %>
        <%= f.text_field   :content , placeholder: 'Add a comment...', class: 'comment-form'%><br>
<% end %>
```

8.4.3 Προσθήκη νέου άρθρου

Άρα η προβολή δηλαδή το View απο το MVC είναι ο παρακάτω κώδικας:

```
(app/views/posts/new.html.erb)

<div class="new-post-container">
   <div class="form-post-container">
   <%= form_for(@post) do |f| %>
   <div class="new-post-flex">
       <%= f.text_field :title, placeholder: 'Whats happening?', class: 'new-post-
title' %>
       <%= f.submit "Post", class: 'input-submit-new-post' %>
       </div>
       <%= f.file_field :image , class: 'new-post-image'%>
<% end %>
  </div>
  <div class="new-post-image">
</div>
```

Η προσθήκη συνδέεται με της δυο function απο το posts controller την new και create

(app/controllers/posts_controller.rb)

```
      def new
            @post = current_user.posts.build
            @users = User.all
            @user = current_user
      end

      def create
            @post = current_user.posts.build(post_params)

            if @post.save
                  redirect_to @post
            else
             render 'new'
            end
      end
```

Η παραπάνω εικόνα είναι η αρχική σελίδα της εφαρμογής μας όταν δεν έχουμε κάνει login εκτός από το animation με τα γράμματα που αλλάζουν δυναμικά μέσα από την βιβλιοθήκη typed.js.

Επίσης έχουμε και δυο links για είσοδο και για εγγραφή στην εφαρμογή.

(app/views/home/index.html.erb)

```
html>
<head>
  <title></title>
  <meta name="description" content="Social Media">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/jpg" href="favicon.jpg">
</head>
<body>
      <div class="home-container">

   <div class="sec-container">
         <div class="header">
           <div class="logo">moments</div>
               <ul class="home-navbar">
                     <li><%= link_to "Sign in",  new_user_session_path %></li>
                     <li><%= link_to "Sign up", new_user_registration_path,
class:'sign' %></li>
               </ul>
          </div>
    <div class="home-wrapper">
          <div class="home-content">
             <div class="text">Share your moments with</div>
           <div class="text-cursor">
                <span class="element"></span>
             </div>
          </div>
        </div>
```

```
        <div class="copy">
         <span class="copy-text">&copy; 2016 MOMENTS</span>  </div>
 </body>
</html>
```

8.5 Τα stylesheets της εφαρμογής και ή responsive προβολή.

Ενα μεγάλο κομμάτι της εφαρμογης ειναι και η css διότι η εφαρμογή είναι σχεδιασμένη για μεγάλες αλλά κυρίως για μικρές συσκεύες.

Στο project δεν χρησιμοποιούμε κάποιο css framework η library είναι όλη καθαρή SASS και ο σχεδιασμός όλου του layout εχει γίνει μέσω του flexbox ενα σχετικά νεο στοιχείο πάνω στην css αλλά πλεόν υποστηρίζεται σχεδόν απο όλους τους browsers.

Το flexbox το χρησιμοποιούμε για να δομήσουμε layouts δηλαδή το πώς θα προβάλουμε το περιεχόμενο.

Πιο συγκεκριμένα τα properties του flexbox χωρίζονται σε δυο βασικές κατηγορίες αυτά που είναι για το parent element και αυτά που είναι για το child element.

Άς δουμε κάποια properties και τις λειτουργίες τους.

Parent

```
display: flex: flex;

flex-direction: row | row-reverse | column | column-reverse;

flex-wrap: nowrap | wrap | wrap-reverse;

justify-content: flex-start | flex-end | center | space-between | space-around |
space-evenly;

align-items: flex-start | flex-end | center | baseline | stretch;
```

Εδω βλέπουμε τα βασικά στοιχεία του γονέα(parent) οπως είναι το flex αρχικά όπου δηλώνοντας ενα div με display flex αυτόματα όλο το div αλλα και τα παιδιά(child) του στοιχλιζονται στο dom με βάση το flexbox και τις λειτουργίες του, ύστερα βλέπουμε και άλλα properties όπως το flex-direction που είναι για την κατεύθηνση των στοιχείων, το flex-wrap που είναι για όταν το view port αλλάξει πχ tablet,smartphone το πώς θα χωρ'εσουν στην σελίδα τα child αυτού του parent.

Children

Τα properties των childrens αναφερονται περισότερο στον τρόπο που θα στοιχιστούν τα στοιχεία στον γονέα.

```
order: <integer>;

flex-grow: <number>;

flex-shrink: <number>;

flex-basis: <length> | auto;
```

Πιο κάτω παρατήθεται ο βασικός κώδικας css που είναι υπευθηνος για το μεγαλύτερο κομμάτι της εφαρμογής.

(app/assets/stylesheets/profile.scss)

```
.header-con {
    display: flex;
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    z-index: 1;
    opacity: 0.97;
    background-color: whitesmoke;
    padding: 17px 5px 5px 5px;
    @media (max-width: 1024px) {
      flex-direction: row;
      justify-content: center;
    }
  .logo-con {
    display: block;
    color: black;
    text-decoration: none;
    font-weight: 700;
    letter-spacing: 4px;
    font-size: 1.62em;
    margin-left: 100px;
    @media (max-width: 667px) {
     display: none;
    }
  }
  .navbar-con {
    margin-top: 5px;
    margin-left: auto;
    margin-right: 100px;
    margin-left: auto;
    @media (max-width: 667px) {
     display:none;
    }
  ul {
    margin: 0;
  }
```

```
 li {
    display: inline-block;
    margin-left: 10px;
  }
  a {
    text-decoration: none;
    color: black;
  }:hover {
    color: $dark-slate;
  }
  .new-post {
    border: 2px solid $dark-slate;
    border-radius: 4px;
    padding: 6px 20px;
    color: $dark-slate;
  }
  .new-post:hover {
    border: 2px solid $black;
    color: $black;
  }
 }
}

.profile-wrapper {
     width: 960px;
     margin: 50px auto;
     padding-top: 50px;
      @media (max-width: 1024px) {
      width: 100%;
    }
  .avatar {
    width: 100%;
    height: 100%;
    border-radius: 4%;
    border: 3px solid $black;
  }


 .avatar-container {
    width: 120px;
    height: 120px;
    border-radius: 7%;
    @media (max-width: 667px) {
      width: 100px;
      height: 100px;
    }
  }
  .info-container {
    margin-top: 10px;
    margin-left: 70px;
    @media (max-width: 667px) {
       margin-top: -2px;
```

```
      margin-left: 22px;
    }
 }
 .profile-name {
   font-size: 1.56em;
   font-weight: 600;
   letter-spacing: 2px;
   color: $black;
    @media (max-width: 667px) {
      font-size: 1.06em;
    }
 }
 .profile-username {
   margin-top: 12.5px;
   font-size: 1em;
   color: $dark-slate;
   @media (max-width: 667px) {
      margin-top: 13px;
    }
 }
 .flex-info {
   display: flex;
   margin-bottom: 30px;
   margin-left: 136px;
   @media (max-width: 667px) {
     margin-left: 1px;
    }
 }
.posts-container {
 display: flex;
 flex-wrap: wrap;
 align-content: flex-start;
 justify-content: space-between;
}
.post-image {
 width: 100%;
 height: 100%;
 border-radius: 3px;
}
.post-container {
 position: relative;
 width: 33%;
 height: 300px;
 margin-bottom: 10px;
   @media (max-width: 667px) {
      width: 33%;
      height: 120px;
    }
}
.post-container:hover {
   filter:brightness(0.6);
  -webkit-filter:brightness(0.6);
}
```

```
.follow {
  color: $dark-slate;
  border-radius: 4px;
  padding: 6px 18px;
  border: 1px solid $dark-slate;
  margin-left: 20px;
  margin-top: 10px;
  letter-spacing: 1px;
  font-weight: 500;
  text-decoration: none;
  @media (max-width: 667px) {
     margin-left: 0px;
     padding: 7px 15px;
     letter-spacing: 2px;
     font-size: 1em;
    }
 }
 .following {
  color: $white;
  background-color: $green;
  border-radius: 4px;
  padding: 8px 14px;
  margin-left: 20px;
  letter-spacing: 1px;
  font-weight: 500;
  text-decoration: none;
   @media (max-width: 667px) {
     margin-left: 0px;
     padding: 7px 15px;
     letter-spacing: 2px;
     font-size: 1em;
    }
 }
 .follow-name-flex {
  display: flex;
  flex-direction: row;
  @media (max-width: 667px) {
    flex-direction: column;
  }
 }
 .follow-name-info {
  display: flex;
 }
 .follow-container {
   margin-top: 6px;
    @media (max-width: 667px) {
     margin-top: 10px;
    }
 }
 .followers-following {
   display: flex;
   margin-top: 17px;
   margin-bottom: 7px;
    @media (max-width: 667px) {
```

```
     font-size: 1em;
     margin-top: 10px;
    }
   .following-users {
   }
   .followers-users {
   margin-left: 15px;

   }
 }
 .posts-count {
  margin-top: 13px;
  margin-left: 15px;
  font-weight: 500;
   @media (max-width: 667px) {
     font-size: 1em;
    }
 }
 .info {
   font-weight: 600;
   letter-spacing: 1px;
 }
}
.responsive-navbar {
  display: none;
  @media (max-width: 667px) {
    display: flex;
  }

.responsive-home {
  margin-top: 4px;
  width: 29px;
  height: 29px;

}
.responsive-img {
  margin-left: 35px;
  margin-top: 1px;
  width: 35px;
  height: 35px;
  border-radius: 50%;

}
.responsive-new {
  margin-left: 35px;
  margin-top: 1px;
  width: 37px;
  height: 37px;

}
.responsive-search {
  margin-left: 28px;
  margin-top: 7px;
  display: block;
}
```

```
.responsive-sign-out {
  margin-left: 35px;
  margin-top: 3px;
  width: 30px;
  height: 30px;
 }
}
.dots {
  margin-left: 15px;
  margin-top: -7px;
  display: none;
  @media (max-width: 667px) {
    display: block;
  }
}

.dots-style {
  text-decoration: none;
  color: $dark-slate;
  font-size: 2em;
}
.search-container {
   width: 100%;
   margin-top: 100px;
   display: flex;
   justify-content: center;
   flex-direction: column;
   align-items: center;
}
 .search-user-photo {
  width: 80px;
  height: 80px;
   border-radius: 50%;
}
.search-user-name {
  font-size: 1.87em;
  letter-spacing: 2px;
  font-weight: 700;
  text-decoration: none;
  margin-left: 20px;
  color: $black;
   @media(max-width: 667px) {
    font-size: 1.56em;
```

(app/assets/stylesheets/index.scss)

```
.main-container {
    background-color: $white;
    margin: 0 auto;
  .main-wrapper {
```

```
   margin: 25px auto;
   width: 550px;
   padding-top: 50px;



@media (max-width: 1024px) {
     width: 100%;
   }
 }
 .moment-container {
   margin: 0 auto;
   width: 100%;
   height: 100%;
   background-color: whitesmoke;
   margin-bottom: 10px;
   border-bottom: 2px solid $light-gray;

    @media (max-width: 667px) {
     width: 100%;
     height: 100%;
   }
 }
 .moment-avatar {
   width: 100%;
   height: 100%;
   border-radius: 50%;
 }
 .moment-avatar-container {
   width: 50px;
   height: 50px;
   padding: 5px 5px;
    @media (max-width: 667px) {
     width: 35px;
     height: 35px;
   }
 }
 }

.moment-flex-one {
   display: flex;
   padding-bottom: 4px;
   margin-top: 12px;
 }
 .moment-name {
   margin-top: 16px;
   margin-left: 8px;
   color: $black;
   letter-spacing: 1px;
   padding: 5px 0;
   font-weight: 600;
   text-decoration: none;
   @media (max-width: 667px) {
      font-size: 0.937em;
```

```
      margin-top: 8px;
    }
  }
  .moment-username {
    color: $gray;
    text-decoration: none;
    font-weight: 500;
    margin-top: 12px;
    margin-left: 5px;

  }
  .time-created {
    margin-top: 15px;
    margin-right: 5px;
    margin-left: auto;
    padding: 5px 5px;
    color: $gray;
    @media (max-width: 667px) {
      font-size:  0.812em;
      margin-top: 8px;
    }
  }
  .moment-image {
    width: 100%;
    height: 100%;
  }
  .moment-image-container {
    width: 100%;
    height: 100%;
    margin-top: 1px;
     @media (max-width: 667px) {
      width: 100%;
      height: 100%;
    }
  }
  .comments-count {
    padding: 5px 5px;
    margin-top: 9px;
    margin-left: 12px;
    margin-right: -4px;
  }
  .flex-two {
    display: flex;
    margin-left: -11px;
    @media (max-width: 667px) {
      margin-left: -7px;
      margin-top: 10px;
    }
  }

  .comments-image {
    width: 30px;
    height: 30px;
    margin-top: 8px;
}
```

```
  .like-image {
    width: 18px;
    height: 18px;
    margin-top: 14px;
  }
  .like-count {
    margin-top: 14px;
    margin-right: 7px;
  }


.sign-out-container {
   width: 25px;
   height: 25px;
    .sign-out {
    margin-left: 7px;
    margin-bottom: -7px;
      width: 100%;
      height: 100%;
  }
}
.search-bar {
  margin-left: auto;
  margin-right: -180px;
  @media(max-width: 1024px) {
    display: none;
   }
}
.search {
   width: 130px;
   height: 25px;
   margin-top: 4px;
   background-color: whitesmoke;
   padding: 2px 5px;
   border-radius: 3px;
   border: 1px solid #dcd7d7;
}
.res-search {
   width: 15px;
   height: 15px;
   border-radius: 4px;
   background-color: whitesmoke;
   padding: 2px 5px;
   border: 2px solid black;
}

.search-animation {
  animation: search-width 0.3s ease-in-out forwards;
}

.res-search-animation {
  animation: res-search-width 0.3s ease-in-out forwards;
}

@keyframes search-width {
```

```
  0% {
     width: 130px;
  }
  100% {
     width: 200px;
  }
}

@keyframes res-search-width {
  0% {
     width: 30px;
  }
  100% {
     width: 200px;
  }
}
```

8.6 Ιδέες βελτίωσης της εφαρμογής

Στο τελευταίο κομμάτι της εργασίας και μετά το τέλος της υλοποίησης της εφαρμογής θέλαμε να παραθέσουμε κάποιες ιδέες που θα μπορούσαν να κάνουν την εφαρμογή να γίνει πιο λειτουργική, πιο γρήγορη και με μικρότερο κόστος.

Αρχικά επειδή η τεχνολογία εξελίσσεται διαρκώς και βγαίνουν συνέχεια νέα πράγματα στην αγορά, όταν ξεκινήσαμε να υλοποιούμε την εφαρμογή μας μετά από έρευνα καταλήξαμε στην ruby on rails κυρίως λόγω του μεγάλου community και σε περίπτωση που δυσκολευόμασταν θα μπορούσαμε να βρούμε κάποια λύση αλλά και επειδή έχει γίνει η υλοποίηση πολλών δυνατών προϊόντων και παρεμφερή στην λογική με το δικό μας όπως το GitHub ήταν μια ασφαλής επιλογή.

Εάν όμως ξεκινούσαμε την υλοποίηση του moments αυτήν την χρονική περίοδο σίγουρα θα επιλέγαμε κάτι εντελώς διαφορετικό και θα προσανατολιζόμασταν στον κόσμο της JavaScript και θα χρησιμοποιούσαμε ReactJS ο βασικός λόγος είναι η ταχύτητα σε σχέση με μια rails εφαρμογή η οποιαδήποτε άλλο framework(jango, Laravel etc..) είναι πολύ πιο γρήγορη λόγω του Virtual Dom αλλά και λόγω της αρχιτεκτονικής.

Στο κομμάτι της αποθήκευσης των δεδομένων μας θα χρησιμοποιούσα redux οπού και αυτό είναι υλοποιημένο από μηχανικούς του Facebook και συμπεριφέρεται τέλεια σε μεγάλο όγκο δεδομένων διότι χρησιμοποιείται σε πολλά προσόντα του Facebook.

Στο κομμάτι του design θα χρησιμοποιούσαμε την ίδια λογική με μικρές διορθώσεις και προσθήκες στην εφαρμογή.

Τέλος σίγουρα θα προσθέταμε την δυνατότητα offline mode οπού είναι μια νέα τεχνολογία της google και είναι τα progressive web apps και σου δίνει την αίσθηση μια mobile εφαρμογής υλοποιημένη όμως με τεχνολογίες του web.

Συμπεράσματα

Μετά από 7 Κεφάλαια πάνω στην εφαρμογή Moments μπορούμε να πούμε πως έχουμε αναλύσει πλήρως την δομή και τον τρόπο σκέψης που πρέπει να ακολουθήσει κάποιος για την δημιουργία μια web εφαρμογής.

Έχουμε αναλύσει την αρχιτεκτονική(MVC) που πρέπει να ακολουθήσει κάποιος για τον σχεδιασμό και την υλοποίησης μιας διαδικτυακής εφαρμογής και τον τρόπο που συνδέονται μεταξύ τους.

Επιπλέον έχουμε αναλύσει όλες τις γλώσσες προγραμματισμού, τα libraries και τα framework που χρησιμοποιούμε για την εφαρμογή και έχουμε παραθέσει όλον τον κώδικα με την ανάλογή δομή.

Στο καθαρά κομμάτι της εφαρμογής έχουμε αναλύσει πλήρως κάθε στάδιο της από την σύλληψη της ιδέας μέχρι και την τελική υλοποίηση και έχουμε αναλύσει λεπτομερώς τον κώδικα τόσο στο backend κομμάτι όσο και στο frontend.

Αναλύσαμε πλήρως την εφαρμογή αρχικά από το διάγραμμα που δείχνει πως συνδέονται οι κλάσης μεταξύ τους και στην συνέχεια κάθε σελίδα της εφαρμογής με εικόνα και κώδικα αλλά και σχολιασμό στις λειτουργίες.

Αναφορές

1) The Ruby Programming Language:
http://shop.oreilly.com/product/9780596516178.do

2) Beginning Ruby: From Novice to Professional:
https://www.amazon.com/Beginning-Ruby-Novice-Professional-Experts/dp/1430223634

3) Agile Web Development with Rails 4:
https://www.amazon.com/Agile-Development-Rails-Facets-Ruby/dp/1937785564/ref=sr_1_2?s=books&ie=UTF8&qid=1482844775&sr=1-2&keywords=agile+web+development+with+rails

4) CSS Secrets: Better Solutions to Everyday Web Design Problems :https://www.amazon.com/s/ref=nb_sb_ss_i_1_7?url=search-alias%3Dstripbooks&field-keywords=css+secrets&sprefix=css+sec%2Cstripbooks%2C270&crid=2YKNE2MBXXG7K

5) HTML and CSS: Design and Build Websites https://www.amazon.com/HTML-CSS-Design-Build-Websites/dp/1118008189/ref=sr_1_3?s=books&ie=UTF8&qid=1482844822&sr=1-3&keywords=css+secrets

6) JavaScript: The Good Parts:
https://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742/ref=sr_1_1?s=books&ie=UTF8&qid=1482844913&sr=1-1&keywords=javascript+the+good+parts

7) JavaScript: The Definitive Guide:
https://www.amazon.com/JavaScript-Definitive-Guide-Activate-Guides/dp/0596805527/ref=sr_1_1?s=books&ie=UTF8&qid=1482844963&sr=1-1&keywords=javascript+the+definitive+guide

8) Ruby History:
https://www.ruby-lang.org/en/about/

9) CSS3:
http://lea.verou.me/css3-secrets/#intro

10) Progressive Web Apps:
https://developers.google.com/web/progressive-web-apps/

11) Ruby Differences
https://www.ruby-lang.org/en/documentation/ruby-from-other-languages/

12) Ruby on Rails Contributor
http://rubyonrails.org/community/

13) Web vs Native
https://www.lifewire.com/native-apps-vs-web-apps-2373133

14) Fake News
http://news.stanford.edu/2017/01/18/stanford-study-examines-fake-news-2016-presidential-election/

15) Flexbox
https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox