ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ POKER

ΠΑΝΑΓΟΥ ΑΝΤΩΝΙΑ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

ΚΩΝΣΤΑΝΤΙΝΟΣ ΣΤΕΡΓΙΟΥ - *Επίκουρος Καθηγητής Π.Δ.Μ.*

Κοζάνη, Ιούνιος 2013

. . . . . .

Αντωνία Πανάγου
Τελειόφοιτη Μηχανικός Πληροφορικής και Τηλεπικοινωνιών Π.Δ.Μ.

ΕΙΣΑΓΩΓΗ

Ο αριθμός των παικτών που παίζουν πόκερ στο διαδίκτυο αυξάνεται συνεχώς όπως αυξάνονται και οι διαδικτυακές αίθουσες πόκερ. Το internet έχει διευκολύνει την εκμάθηση του παιχνιδιού πόκερ. Οι online κοινότητες πόκερ είναι ένα εκπληκτικό εργαλείο για οδηγίες και συμβουλές σχετικά με το πόκερ και ταυτόχρονα είναι ένα μέσο για γνωριμία καινούργιων φίλων. Το διαδικτυακό πόκερ βρίσκεται ακόμα στα πρώτα του βήματα όσον αφορά τον αριθμό των παικτών και την τεχνολογία των διαδικτυακών αιθουσών πόκερ ενώ η στρατηγική για το διαδικτυακό πόκερ εξελίσσεται με γρήγορους ρυθμούς.

Για πρώτη φορά, η Ελλάδα βρίσκεται στην κορυφή των εξελίξεων στο χώρο του διαδικτυακού πόκερ , αυτό αφ' ενός θα βοηθήσει την χώρα να δημιουργήσει επιπλέον έσοδα και αφ' ετέρου ευνοεί πολύ την πρόοδο και την επέκταση του επαγγελματικού πόκερ στην Ελλάδα.

Η παρούσα διπλωματική εργασία πραγματεύεται την ανάπτυξη λογισμικού για poker με όνομα “THE FUNNY POKER”. Ειδικότερα , κύριος στόχος της είναι να κατασκευαστεί ένα λογισμικό στο οποίο θα μπορούν να παίζουν ,“ζωντανά” ένας παίκτης εναντίον του υπολογιστή. Η μορφή πόκερ που παίζουν είναι το five cards draw. Σύμφωνα με τους κανόνες σε αυτό το παιχνίδι μοιράζονται από πέντε φύλλα στο κάθε συμμετέχοντα .Έπειτα, και οι δύο έχουν το δικαίωμα να αλλάξουν τα τρία από αυτά τα φύλλα, προκειμένου να επιτύχουν καλύτερο συνδυασμό φύλλων και στο τέλος ποντάρουν στο φύλλο τους και αναμετρούνται.

Το Funny Poker προσφέρει την δυνατότητα σε έναν παίκτη να παίξει εναντίον της μηχανής. Ο παίκτης κάνει login στην σελίδα και αυτόματα έχει πρόσβαση στο παιχνίδι funny poker και στο πορτοφόλι του. Το πορτοφόλι του είναι ένα εργαλείο με το οποίο μπορεί να μεταφέρει χρήματα από μια πιστωτική κάρτα στο λογαριασμό του και στην συνέχεια να τα ποντάρει στο παιχνίδι. Το παιχνίδι funny poker ξεκινάει μοιράζοντας από πέντε φύλλα σε κάθε συμμετέχοντα και ο πραγματικός παίκτης (όχι η μηχανή) έχει την δυνατότητα να ποντάρει δύο φορές, μια φορά πριν αλλάξει τα τρία φύλλα και μια φορά αφού τα αλλάξει. Η μηχανή δέχεται οποιοδήποτε ποντάρισμα και με την σειρά της αλλάζει και αυτή από τρία φύλλα. Στο τέλος αναμετριούνται και κερδίζει ο συμμετέχοντας του παιχνιδιού που έχει τον καλύτερο συνδυασμό φύλλων μετά τις αλλαγές.

Για την ανάπτυξη του λογισμικού αυτού χρησιμοποιήθηκε κυρίως ως γλώσσα προγραμματισμού η javascript για την δημιουργία σεναρίων. Επιπλέον, το περιβάλλον στο οποίο λειτουργεί είναι το meteor μια πλατφόρμα open-source για την δημιουργία web εφαρμογών. Το Meteor προσφέρει μια σύνθεση δυνατοτήτων για διαχείριση και λειτουργία βάσεως δεδομένων, application server, και περιβάλλον λειτουργίας UI framework. Η βάση δεδομένων είναι η mongodb, ο application server βασίζεται στον node.js και το UI σε main stream HTLM5 frameworks όπως bootstrap, underscore.js, jquery κα.

ΠΕΡΙΛΗΨΗ

Σε αυτή την διπλωματική εργασία δημιουργείται εξ' ολοκλήρου μια εφαρμογή με την οποία ο χρήστης μπορεί να παίξει five cards draw poker εναντίον της μηχανής. Στο παιχνίδι αυτό μοιράζονται από πέντε φύλλα σε κάθε παίκτη, βλέποντας αυτά τα φύλλα ο πραγματικός παίκτης ποντάρει κάποιο ποσό και η μηχανή δέχεται το ποντάρισμα αυτόματα. Έπειτα και οι δύο παίκτες έχουν την δυνατότητα να αλλάξουν μέχρι τρία από τα φύλλα τους και αυτά να αντικατασταθούν με τυχαία από την τράπουλα φύλλα. Μετά τις αλλαγές αυτές ο πραγματικός παίκτης ξανά ποντάρει στο φύλλο του για να δει τα φύλλα τις μηχανής και να κριθεί ο νικητής. Σκοπός του παιχνιδιού είναι μετά τις αλλαγές αυτές να επιτύχουν ένα καλό συνδυασμό ώστε να κερδίσουν την παρτίδα. Στο τέλος λοιπόν του παιχνιδιού ανοίγουν τα φύλλα και αυτός με τον καλύτερο συνδυασμό κερδίζει το πόσο που παίχτηκε στο πρώτο και στο δεύτερο ποντάρισμα.

Για την δημιουργία αυτής της εφαρμογής χρησιμοποιήθηκε το meteor. Το Meteor είναι ένα εξαιρετικά αποτελεσματικό περιβάλλον για την οικοδόμηση εφαρμογών και προσφέρει μια σύνθεση δυνατοτήτων για διαχείριση και λειτουργία βάσεως δεδομένων, application server, και περιβάλλον λειτουργίας UI framework. Το Meteor λειτουργεί σύμφωνα με το model-view-controller(MVC). Το MVC μπορεί να διαιρέσει ένα συστατικό σε τρία λογικά μέρη: μοντέλο (model), αναπαράσταση (view) και διαχείριση (controller) καθιστώντας ευκολότερη την διαδικασία τροποποίησης κάθε μέρους.

Στα κεφάλαια τις διπλωματικής αναλύονται το μοντέλο (model), η αναπαράσταση (view) και η διαχείριση (controller) των κομματιών στα οποία έχει χωριστεί η εφαρμογή αυτή. Ειδικότερα, η εφαρμογή χωρίζεται σε τέσσερα μέρη :

I. *Cards*: Όπως υποδηλώνει και το όνομα πρόκειται για την μοντελοποίηση των φύλλων της τράπουλας.
II. *Game*: Δηλώσεις για την λειτουργία της συγκεκριμένης παραλλαγής του poker.
III. *Poker*: Οι κανόνες του poker , η δήλωση των ιεραρχιών των συνδυασμών.
IV. *Main:* Δήλωση των συστατικών μερών της ιστοσελίδας.

Όλα τα παραπάνω θα αναλυθούν λεπτομερώς στα επιμέρους μέρη της εργασίας αυτής.

ΕΥΧΑΡΙΣΤΙΕΣ

Στο σημείο αυτό θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες σε όλους όσους συνέβαλαν ώστε να έρθει εις πέρας η παρούσα προπτυχιακή διπλωματική εργασία.

Αισθάνομαι την ανάγκη να ευχαριστήσω θερμά τον διευθυντή της openbet κ. Δημήτρη Λιβά για την πολύτιμη βοήθεια του, καθώς και τον Επίκουρο Καθηγητή και Επιβλέποντα της εργασίας αυτής κ. Κωνσταντίνο Στεργίου για την ευκαιρία που μου έδωσε να εργαστώ σε ένα σύγχρονο αντικείμενο.

Τέλος, ευχαριστώ την οικογένειά μου για την αγάπη, την συμπαράσταση, την κατανόηση και ανοχή της σε όλη τη διάρκεια της φοιτητικής μου διαδρομής.

Κοζάνη, Ιούνιος 2013

Αντωνία Πανάγου

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1: ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

1.1 Το Poker σήμερα

Τα τελευταία περίπου 30 χρόνια το πόκερ έχει αυξήσει σημαντικά την διείσδυσή του στην κοινωνία, τόσο στο εξωτερικό, όσο και στη χώρα μας. Μπορεί παραδοσιακά στη χώρα μας να παίζονταν σχεδόν σε καθολικό βαθμό η πόκα, αλλά σήμερα τα πράγματα έχουν αλλάξει και το πόκερ αποτελεί τη νούμερο ένα επιλογή των περισσοτέρων παικτών. Το ερώτημα που προκύπτει απ' τα παραπάνω δεδομένα είναι το που οφείλεται αυτό το αυξημένο ενδιαφέρον για το πόκερ.

Η πρώτη σημαντική παράμετρος αφορά τον τρόπο με τον οποίο διαμορφώνονται τάσεις στην παγκόσμια κουλτούρα. Αυτός ο τρόπος δεν είναι άλλος από τον κινηματογράφο και τη τηλεόραση, και πιο συγκεκριμένα τις αγγλόφωνες ταινίες και τηλεοπτικές σειρές που τις βλέπουν δισεκατομμύρια άνθρωποι στην υφήλιο. Δεν είναι τυχαίο ότι τις τελευταίες δεκαετίες παρατηρείται αύξηση της παρουσίας του πόκερ στο τηλεοπτικό και κινηματογραφικό περιεχόμενο, αλλά και αύξηση των αναφορών στο πόκερ.

Η δεύτερη σημαντικότερη παράμετρος, και μεταγενέστερη σε χρονολογική σειρά, είναι το διαδίκτυο και η εξάπλωσή του. Ενώ στην εποχή προ διαδικτύου το στήσιμο μιας παρτίδας πόκας σε ένα Ελληνικό σπίτι ήταν αρκετά πολύπλοκη υπόθεση (χρειάζονταν άτομα που γνώριζαν το παιχνίδι, μέρος και βολικός χρόνος συνάντησης, διαθεσιμότητα των παικτών), σήμερα με λίγα κλικ ο καθένας μπορεί να παίξει ανά πάσα στιγμή δωρεάν ποκερ με αντιπάλους απ' όλο το κόσμο, εύκολα και γρήγορα .Αυτή η ευκολία στη πρόσβαση δεν άργησε να φέρει φαινόμενα ραγδαίας αύξησης στο παιχνίδι του poker, είτε μέσω κοινωνικών δικτύων, είτε μέσα από εξειδικευμένες αίθουσες πόκερ .

Σήμερα υπάρχουν κυριολεκτικά χιλιάδες πλατφόρμες πόκερ μέσω διαδικτύου με τον παίκτη να έχει ένα σημαντικότατο εύρος επιλογών. Ως συνέπεια αυτού, υπάρχει και μεγάλη εμπορική προώθηση του πόκερ από τις διάφορες πλατφόρμες η οποία διευρύνει ακόμα περισσότερο την ήδη επιταχυνόμενη ανάπτυξη του παιχνιδιού. Για τον Έλληνα παίκτη τόσο η τηλεόραση όσο και το διαδίκτυο εισήγαγαν κάποιες νέες παραμέτρους στο παιχνίδι, όπως για παράδειγμα τα τουρνουά πόκερ τα οποία, σε γενικές γραμμές, του ήταν άγνωστα.

1.2 Κανόνες παιχνιδιού

Δεν μπορούμε εύκολα να εξηγήσουμε με ένα και μόνο ορισμό τι είναι το πόκερ. Υπάρχουν πάρα πολλές παραλλαγές του παιχνιδιού αν και η πλειοψηφία τους έχει περίπου τους ίδιους κανόνες. Πόκερ μπορεί να παίξει κάποιος είτε σε κάποιο poker room σε καζίνο, είτε στο σπίτι με τους φίλους είτε σε κάποιον online poker room.

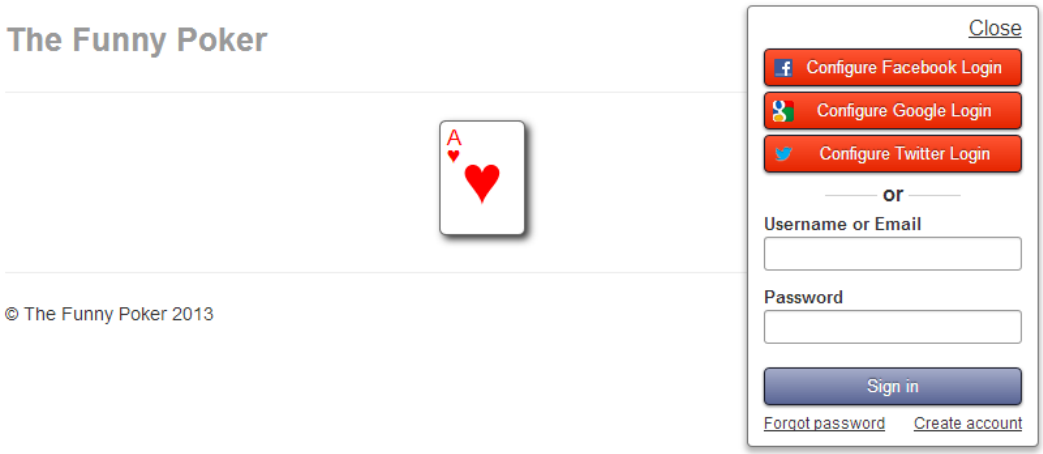
Στα live poker room τα παιχνίδια που παίζονται χωρίζονται σε τρεις κατηγορίες στα stud, draw και flop. Ανάμεσα στις πιο δημοφιλείς παραλλαγές του πόκερ είναι το Texas Hold 'Em, Seven και Five Card Stud, Omaha High, Omaha High/Low και το Razz. Από την άλλη, στα παιχνίδια που παίζονται στο σπίτι υπάρχουν πάρα πολλές παραλλαγές (η γνωστή "πόκα"). Σήμερα η πιο δημοφιλής παραλλαγή πόκερ στον κόσμο είναι το Texas Hold 'Em. Αυτό χάριν στην τηλεοπτική κάλυψη η οποία πρόβαλλε την απλότητα του παιχνιδιού.

Σχεδόν όλα τα παιχνίδια πόκερ παίζονται με μία τράπουλα που αποτελείται από 52 φύλλα. Παρά κάτω προβάλλεται η ιεραρχία των χεριών πόκερ ξεκινώντας από το πιο ισχυρό και καταλήγοντας στο πιο αδύναμο:

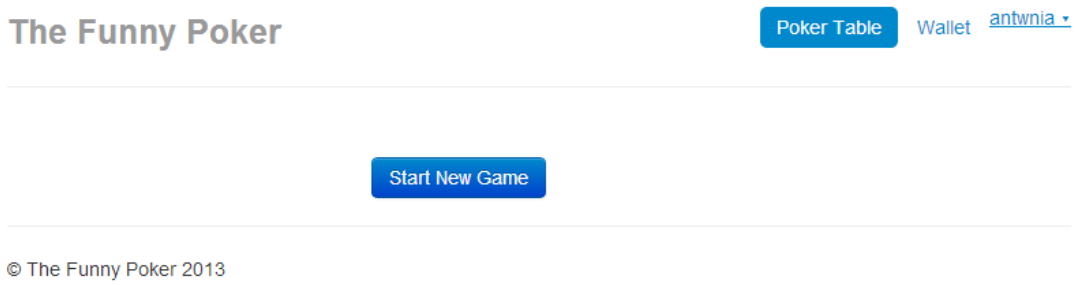
Παράδειγμα	Κατάταξη Συνδυασμών απο το Υψηλότερο στο Χαμηλότερο
A♥K♥Q♥J♥T♥ Φλος Ρουαγιάλ, Υψηλό Ασσο Κούπα	**Φλος Ρουαγιάλ** *"Royal"* Τα πέντε υψηλότερα φύλλα σε χρώμα. Το Φλος Ρουαγιάλ βασικά είναι Κέντα Φλος υψηλού Ασσου.
A♥2♥3♥4♥5♥ Κέντα Φλος, Πέντε υψηλό	**Κέντα Φλος** Οποιαδήποτε πέντε συνεχόμενα φύλλα του ίδιου χρώματος. Το παράδειγμα είναι πέντε υψηλό Κέντα Φλος. (Μόνο ο Ασσος μπορεί να είναι υψηλός ή χαμηλός για Κέντα.)
K♣K♦K♥K♠8♠ 4 Ρηγάδες	**Καρέ** *"Quads"* Τέσσερα όμοια φύλλα, αριθμού ή φιγούρας (πχ 4 Ρηγάδες)
Q♣Q♦Q♥K♣K♦ Φουλ, Ντάμες πάνω απο Ρηγάδες	**Φουλ** *"Full House"* Τριπλό μιας κατάταξης καθώς και ζευγάρι απο άλλο. Το παράδειγμα είναι Ντάμα over Ρηγάδες, το οποίο κερδίζει Βαλέ over Ασσους λόγω του τριπλού.
K♥2♥3♥7♥10♥ Φλος, Υψηλός Ρήγας	**Φλος** Οποιαδήποτε πέντε φύλλα του ίδιου χρώματος. (αν είναι συνεχόμενα έχετε Κέντα Φλος.)
A♣K♦Q♥J♠10♠ Κέντα, Υψηλός Άσσος	**Κέντα** *"Wheel"* Πέντε συνεχόμενα φύλλα του ανεξαρτήτου χρώματος. Ο Ασσος μπορεί να είναι ψηλά (δίπλα σε Ρήγα) η χαμηλά (δίπλα σε 2) αλλά όχι και στα δυο ταυτόχρονα.
5♣5♦5♥K♣3♦ Τρια Πέντε	**Τρία Όμοια Φύλλα** *"Trips", "Triplets", "Set"* Τρια όμοια φύλλα αριθμού ή φιγούρας. (αν τα άλλα δυο έκαναν ζευγάρι θα είχατε Φουλ.)
7♣7♦4♥4♣3♦ Δυο Ζεύγη, Επτά και Τέσσερα	**Δυο Ζεύγη** Οποιοδήποτε Ζευγάρι μιας κατάταξης καθώς και ένα ζευγάρι άλλης κατάταξης. Αν δυο χέρια έχουν το ίδιο υψηλό ζευγάρι, το δεύτερο ζεύγαρι αποφασίζει το νικητή. Αν υπάρξει ισοπαλία τα απομείναντα υψηλά φύλλα αποφασίζουν.
10♣10♦5♥K♣3♦ Ενα Ζευγάρι Δεκάρια	**Ζευγάρι** Οποιαδήποτε δυο φύλλα ίδιας κατάταξης. Οταν δυο χέρια έχουν το ίδιο ζευγάρι, το υψηλό φύλλο που έχει απομείνει αποφασίζει.
J♣5♦9♥K♣3♦ Υψηλός Ρήγας	**Υψηλό Φύλλο** Αν δεν επιτευχθεί καμμία κατάταξη χαρτιών το υψηλότερο φύλλο κερδίζει. Αν δυο χέρια έχουν το ίδιο υψηλό χαρτί τότε τα υψηλά φύλλα που παραμένουν αποφασίζουν

1.3 Περιγραφή του FUNNY POKER

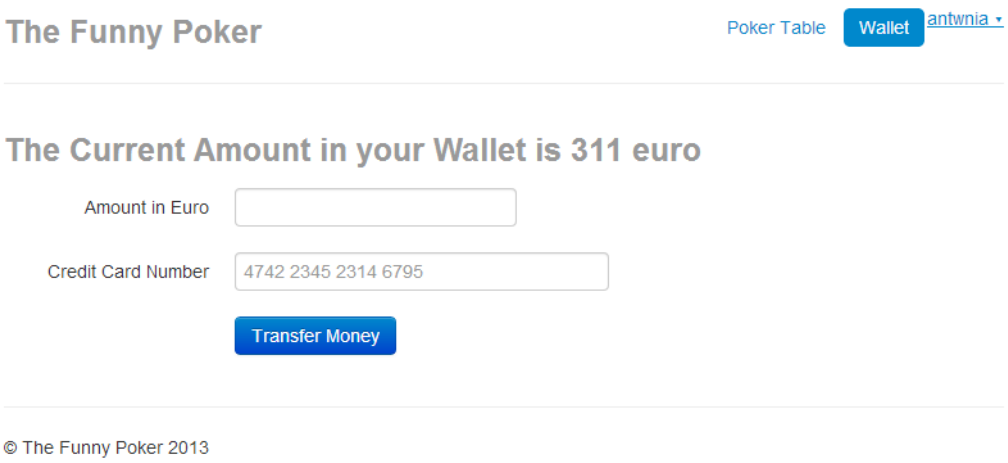
Στην συγκεκριμένη εργασία θα ασχοληθούμε με ένα παιχνίδι μεταξύ 2 αντιπάλων (μηχανή - επισκέπτης), μία τράπουλα 52 φύλλων (χωρίς joker) και μάρκες αξίας 0€ 10€, 20€ , 50€ και 200€.
Για να ξεκινήσει ο παίκτης θα πρέπει να κάνει εγγραφή στην σελίδα . Ο παίκτης μπορεί επίσης να χρησιμοποιήσει την εφαρμογή από τον λογαριασμό του στο facebook , στην google και στο twitter.



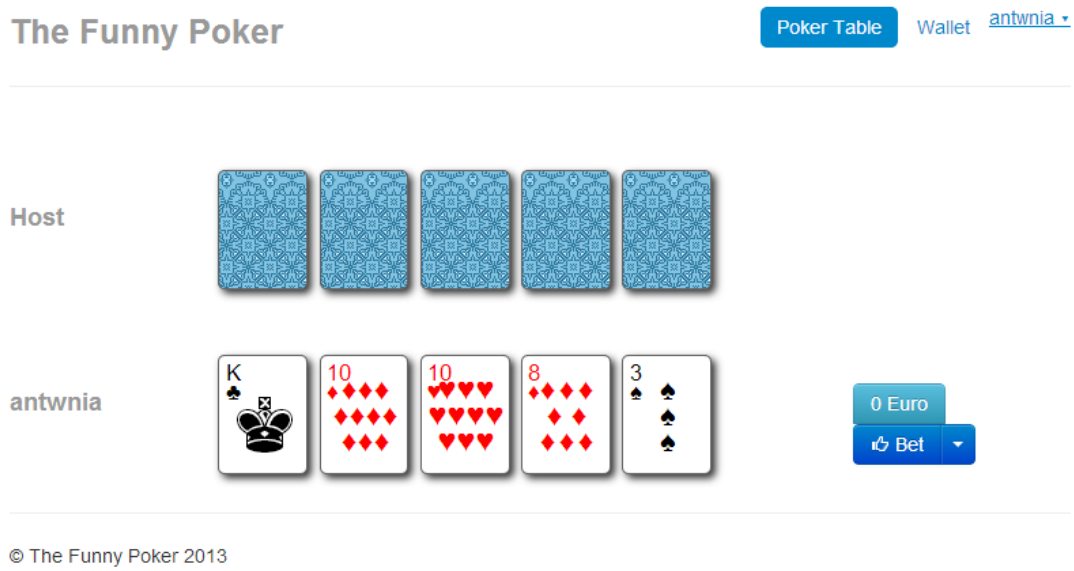
Αφού δημιουργήσει τον λογαριασμό του θα εισέλθει στην σελίδα που θα του επιτρέψει να ξεκινήσει το παιχνίδι.



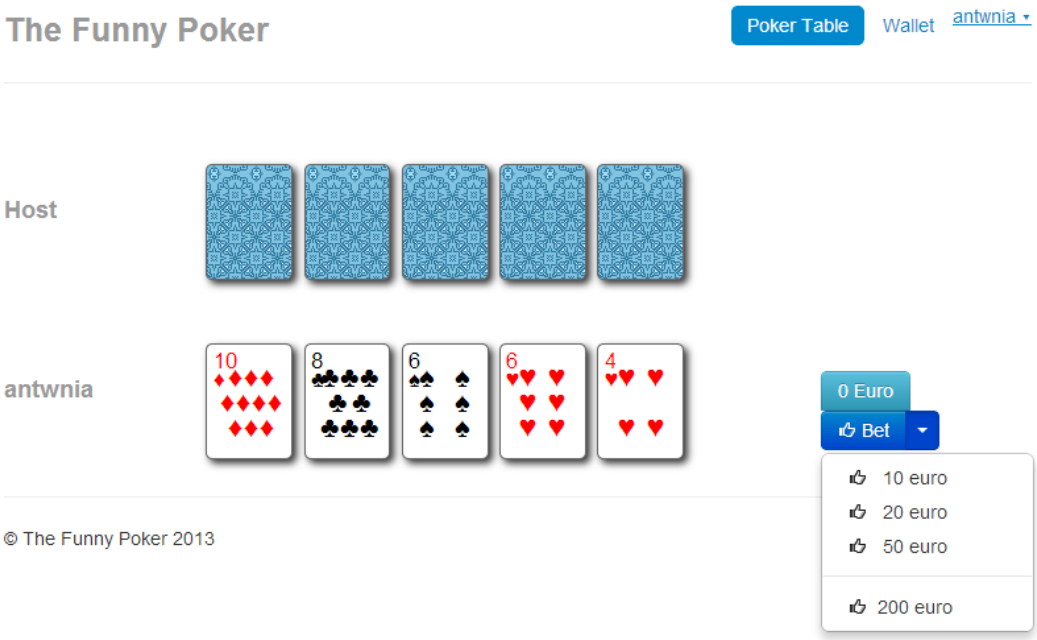
Πάνω δεξιά υπάρχει ένα εικονικό πορτοφόλι. Ο εκάστοτε χρήστης έχει το δικό του πορτοφόλι στο οποίο μπορεί να μεταφέρει χρήματα μέσα πιστωτικής κάρτας. Εδώ χρησιμοποιούμε μια εικονική πιστωτική κάρτα οπού μπορούμε να μεταφέρουμε ότι ποσό επιθυμούμε. Όλοι οι παίκτες την πρώτη φορά που χρησιμοποιούν την εφαρμογή διαθέτουν αρχικά στο wallet τους το ποσό των 300€. Στο οποίο ποσό μπορούν να προσθέσουν χρήματα από μια εικονική πιστωτική κάρτα .



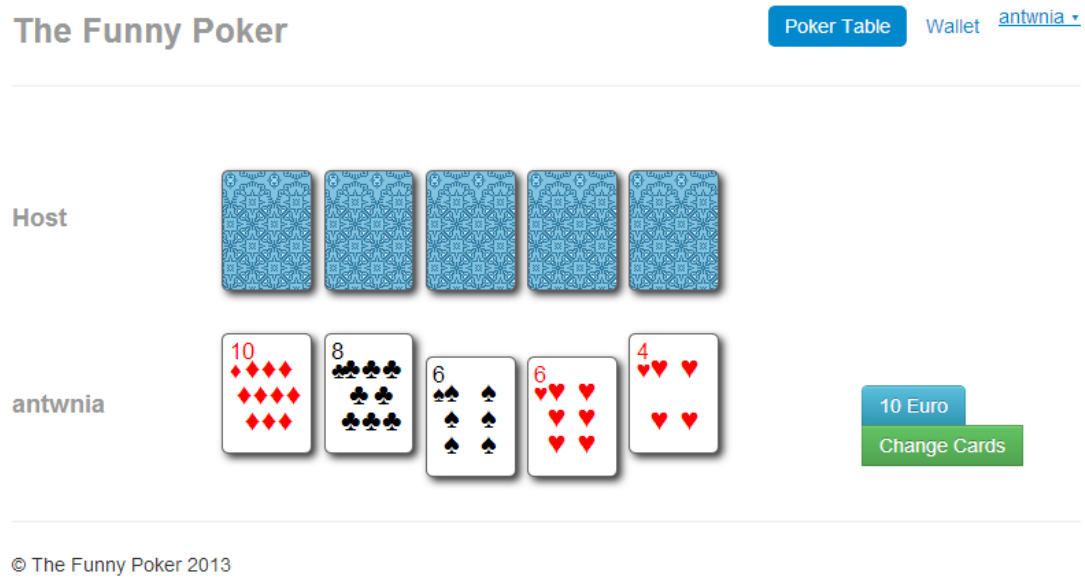
Στη κατηγορία wallet φαίνεται τι ποσό διαθέτει ο εκάστοτε παίκτης. Έστω, ότι έχουμε όπως φαίνεται και στην παραπάνω εικόνα 311€. Πηγαίνοντας πάλι στην κατηγορία Poker table ξεκινάμε το παιχνίδι.

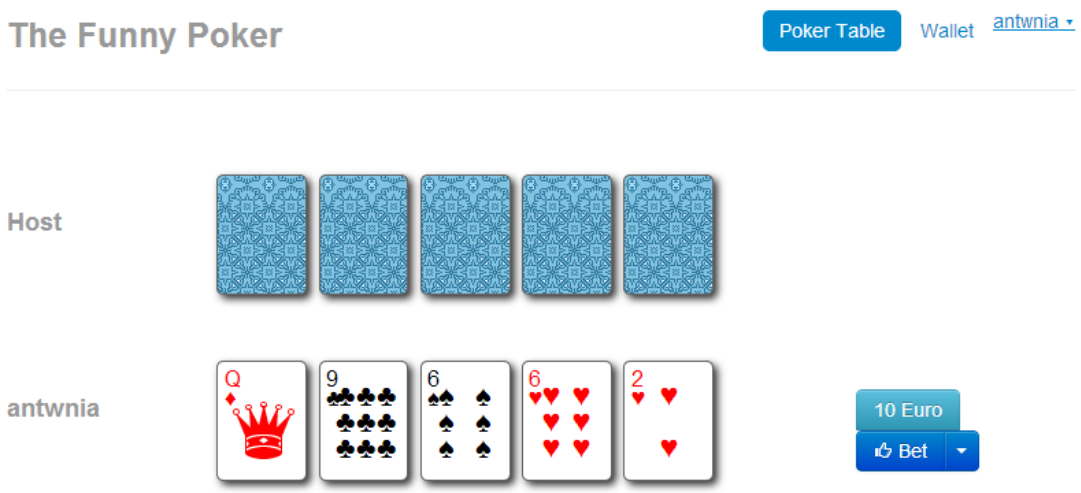


Στην πάνω γραμμή εμφανίζονται τα κρυφά φύλλα του υπολογιστή και στην κάτω τα φύλλα του παίκτη. Μοιράζονται λοιπόν από πέντε χαρτιά σε καθένα από τους συμμετέχοντες του παιχνιδιού. Ο πραγματικός παίκτης βλέποντας τα πέντε φύλλα του επιλέγει ποια θέλει να κρατήσει και ποια θέλει να αλλάξει . Εχει την δυνατότητα να αλλάξει τρία φυλλα. Ωστόσο μπορεί να αλλάξει και τέσσερα μόνο αν διαθέτει άσσο. Τα φύλλα που θέλει να αλλάξει τα επιλέγει με το ποντίκι (με μονό αριστερό click) και ποντάρει στο φύλλο του πριν έρθουν οι αλλαγές. Από το κουμπί bet ποντάρει 0€ ή 10€ ή 20€ ή 50€ ή 200€.



Εστώ ότι ποντάρει 10 ευρώ και επιλέγει να κρατήσει το ζεύγος του 6 και να αλλάξει τα υπόλοιπα τρία φύλλα 10, 8 και 4. Επιλέγει με click το 10, το 8 και το 4 και πατώντας το "Change Cards" τα φύλλα αυτά αντικαθιστούνται.





Ο παίκτης παραλαμβάνει τα νέα του φύλλα. Παράλληλα ο υπολογιστής σύμφωνα με τον έξυπνο αλγόριθμο που θα αναλύσουμε αργότερα αλλάζει και αυτός φύλλα . Ο παίκτης , τώρα, πρέπει να ξανά ποντάρει στο νέο του φύλλο από το bet. Έστω, ότι ποντάρει άλλα 20 ευρώ για το ζεύγος του 6 που έχει. Αυτόματα ανοίγουν τα φύλλα του υπολογιστή και ο καλύτερος συνδυασμός φύλων κερδίζει το ποσό των 30 ευρώ.

Όπως βλέπουμε ο χρήστης antwnia κερδίζει την παρτίδα αφού ο υπολογιστής έχει high cart ντάμα . Αυτόματα τα χρήματα αυτά πηγαίνουν στο wallet του. Έπειτα το παιχνίδι μπορεί να ξεκινήσει από την αρχή ξανά μοιράζοντας από 5 φύλλα σε κάθε μέλος του παιχνιδιού.

ΚΕΦΑΛΑΙΟ 2 : ΤΕΧΝΟΛΟΓΙΕΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ

2.1. meteor

Το Meteor είναι ένα εξαιρετικά αποτελεσματικό περιβάλλον για την οικοδόμηση εφαρμογών. Το Meteor προσφέρει μια σύνθεση δυνατοτήτων για διαχείριση και λειτουργία βάσεως δεδομένων, application server, και περιβάλλον λειτουργίας UI framework. Η βάση δεδομένων είναι η mongodb, ο application server βασίζεται στον node.js και το UI σε main stream HTLM5 frameworks όπως bootstrap, underscore.js, jquery κα.

- *Επτά Ιδιότητες της Meteor*

1.) *Data on the Wire*. Don't send HTML over the network. Send data and let the client decide how to render it.
2.) *One Language.* Write both the client and the server parts of your interface in JavaScript.
3.) *Database Everywhere*. Use the same transparent API to access your database from the client or the server.
4.) *Latency Compensation*. On the client, use prefetching and model simulation to make it look like you have a zero-latency connection to the database.
5.) *Full Stack Reactivity*. Make realtime the default. All layers, from database to template, should make an event-driven interface available.
6.) *Embrace the Ecosystem*. Meteor is open source and integrates, rather than replaces, existing open source tools and frameworks.
7.) *Simplicity Equals Productivity*. The best way to make something seem simple is to have it actually *be* simple. Accomplish this through clean, classically beautiful APIs.

Το δομικό κομμάτι της εφαρμογής

Μια meteor εφαρμογή είναι ένα μείγμα JavaScript που τρέχει μέσα στον client με ένα JavaScript που τρέχει στον meteor server μέσω του node.js μαζί με όλα τα τα HTML fragments και όλα τα απαραίτητα CSS. Η Meteor αυτοματοποιεί τα πακέτα και τη διαβίβαση αυτών των διαφορετικών στοιχείων και είναι αρκετά ευέλικτη για το πώς θα επιλέξει να διαμορφώσει εκείνα τα συστατικά στο δέντρο των αρχείων .

Ο server τρέχει σε JavaScript. Η meteor συγκεντρώνει όλα τα JavaScript αρχεία από τους υποκαταλόγους του server και του client και τα φορτώνει στο Node.js. Το meteor εξετάζει ξεχωριστά τα αρχεία του server και τα δημόσια αρχεία από τα αρχεία του client. Μορφοποιεί αυτό το πακέτο και το σερβίρει σε νέο client . Μπορεί να χρησιμοποιήσει ο χρήστης ένα JavaScript αρχείο μόνο του στην εφαρμογή ή να δημιουργήσει ένα δέντρο με ξεχωριστά αρχεία. Τα αρχεία έξω από τους υποκαταλόγους του client , του server και του tests τρέχουν ταυτόχρονα και στον client και στον server.

Το Meteor παρέχει τις μεταβλητές isClient και isServer έτσι ώστε ο κωδικός να μπορεί να αλλάξει τη συμπεριφορά του ανάλογα με το αν τρέχει στον client ή στο server. Κάθε ευαίσθητος κώδικα που δεν πρέπει να φαίνεται στον client, όπως είναι ο κώδικας που περιέχει τους κωδικούς πρόσβασης ή τους μηχανισμούς ελέγχου ταυτότητας, θα πρέπει να κρατιούνται στον φάκελο του server.

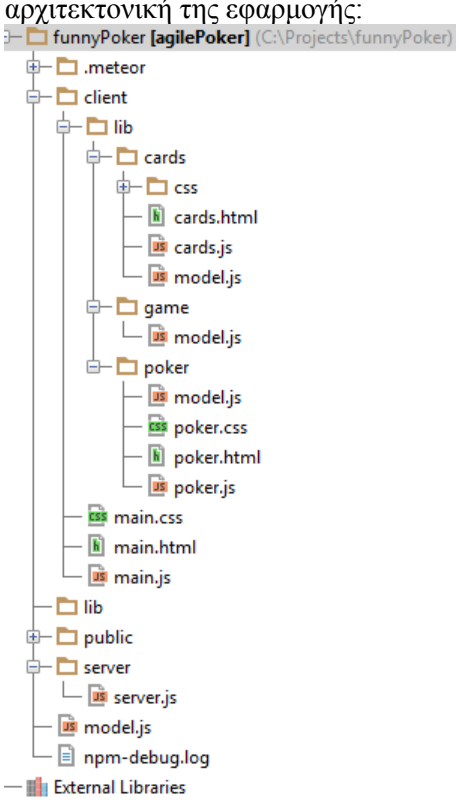
Τα αρχεία CSS συγκεντρώνονται μαζί, καθώς ο client θα πάρει ένα πακέτο με όλα τα CSS αρχεία του δέντρου. Στη λειτουργία της ανάπτυξης, τα JavaScript και CSS αρχεία αποστέλλονται ξεχωριστά για να κάνουν ευκολότερη αποσφαλμάτωση.

Τα HTML αρχεία στο Meteor αντιμετωπίζονται αρκετά διαφορετικά από την πλευρά server. Το Meteor σαρώνει όλα τα αρχεία HTML στον κατάλογο για τρία βασικά στοιχεία: <head>, <body> και <template>. Τα τμήματα head και body ενώνονται σε ένα ενιαίο head και body, τα οποία διαβιβάζονται στον πελάτη (client) κατά την αρχική φόρτωση της σελίδας. Το τμήμα template μετατρέπεται σε Javascript functions για να μεταφέρονται ευκολότερα στον πελάτη.

Στο Meteor, ο client και ο server μοιράζονται το ίδιο API βάσης δεδομένων. Ο ίδιος ακριβώς κώδικας της εφαρμογής μπορεί συχνά να τρέξει και στα δύο μέρη. Αλλά όταν ο κώδικας εκτελείται στον server έχει άμεση πρόσβαση στη βάση δεδομένων, ενώ στον client δεν συμβαίνει αυτό. Η ασφάλεια δεδομένων έγκειται σε αυτή την διάκριση που κάνει η Meteor.

Σήμερα, οι περισσότερες εφαρμογές του Meteor χρησιμοποιούν την MongoDB ως βάση δεδομένων τους, διότι μπορεί να την υποστηρίξει καλύτερα από οποιαδήποτε άλλη. Την λειτουργία της MongoDB θα την περιγράψουμε στο επόμενο υποκεφάλαιο.

Στην επόμενη εικόνα φαίνεται για την συγκεκριμένη εφαρμογή τα αρχεία του meteor και η αρχιτεκτονική της εφαρμογής:



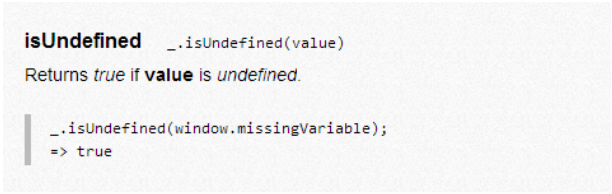
2.2 Underscore.js

Η Underscore είναι βοηθητική βιβλιοθήκη για JavaScript που παρέχει ένα μεγάλο μέρος λειτουργικής υποστήριξης προγραμματισμού αλλά χωρίς να ενσωματώνετε σε αντικείμενα JavaScript. Στην σελίδα http://underscorejs.org/# παρουσιάζονται όλες η διαθέσιμες εντολές της underscore , ενώ παρακάτω θα παρουσιάσουμε σε αυτό το υποκεφάλαιο με εικόνες την λειτουργία των εντολών που χρησιμοποιήθηκαν σε αυτήν την διπλωματική εργασία.

isUndefined `_.isUndefined(value)`

Returns *true* if **value** is *undefined*.

```
_.isUndefined(window.missingVariable);
=> true
```

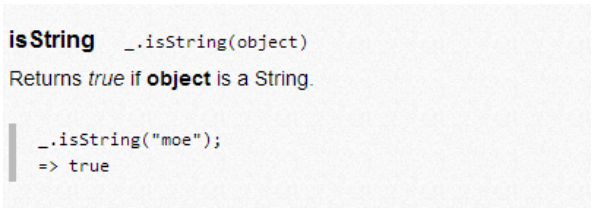


Εικόνα 2.2.1 Λειτουργία της _.isUndefined(value).

isString `_.isString(object)`

Returns *true* if **object** is a String.

```
_.isString("moe");
=> true
```

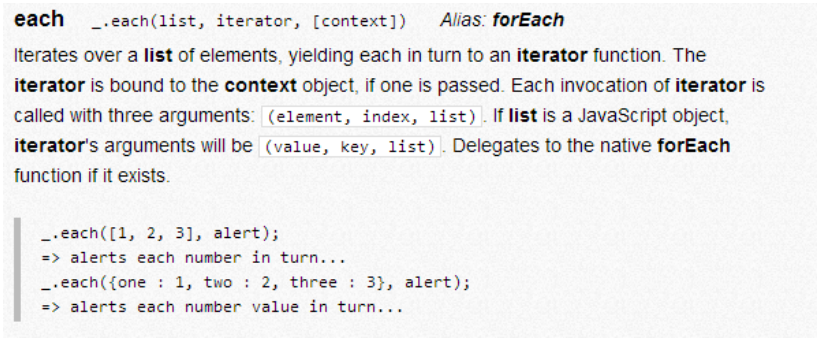


Εικόνα2.2.2 Λειτουργία της _.isString(object)

each `_.each(list, iterator, [context])` *Alias:* ***forEach***

Iterates over a **list** of elements, yielding each in turn to an **iterator** function. The **iterator** is bound to the **context** object, if one is passed. Each invocation of **iterator** is called with three arguments: `(element, index, list)`. If **list** is a JavaScript object, **iterator**'s arguments will be `(value, key, list)`. Delegates to the native **forEach** function if it exists.

```
_.each([1, 2, 3], alert);
=> alerts each number in turn...
_.each({one : 1, two : 2, three : 3}, alert);
=> alerts each number value in turn...
```

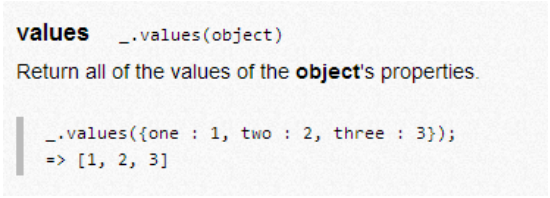


Εικόνα2.2.3 Λειτουργία της _.each(list, iterator, [context])

values `_.values(object)`

Return all of the values of the **object**'s properties.

```
_.values({one : 1, two : 2, three : 3});
=> [1, 2, 3]
```

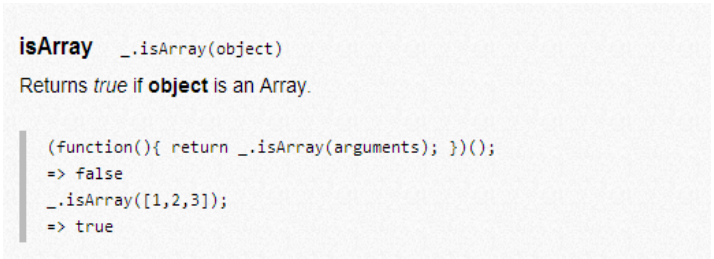


Εικόνα2.2.4 Λειτουργία της _.values(object)

isArray `_.isArray(object)`

Returns *true* if **object** is an Array.

```
(function(){ return _.isArray(arguments); })();
=> false
_.isArray([1,2,3]);
=> true
```



Εικόνα2.2.5 Λειτουργία της _.isArray(object)

shuffle `_.shuffle(list)`

Returns a shuffled copy of the **list**, using a version of the Fisher-Yates shuffle.

```
_.shuffle([1, 2, 3, 4, 5, 6]);
=> [4, 1, 6, 3, 5, 2]
```

Εικόνα2.2.6 Λειτουργία της _.shuffle(list)

first `_.first(array, [n])` *Alias:* ***head***, ***take***

Returns the first element of an **array**. Passing **n** will return the first **n** elements of the array.

```
_.first([5, 4, 3, 2, 1]);
=> 5
```

Εικόνα2.2.7 Λειτουργία της _.first(array, [n])

extend `_.extend(destination, *sources)`

Copy all of the properties in the **source** objects over to the **destination** object, and return the **destination** object. It's in-order, so the last source will override properties of the same name in previous arguments.

```
_.extend({name : 'moe'}, {age : 50});
=> {name : 'moe', age : 50}
```

Εικόνα2.2.8 Λειτουργία της _.extend(destination, *sources)

findWhere `_.findWhere(list, properties)`

Looks through the **list** and returns the *first* value that matches all of the key-value pairs listed in **properties**.

```
_.findWhere(publicServicePulitzers, {newsroom: "The New York Times"});
=> {year: 1918, newsroom: "The New York Times",
  reason: "For its public service in publishing in full so many official reports,
  documents and speeches by European statesmen relating to the progress and
  conduct of the war."}
```

Εικόνα2.2.9 Λειτουργία της _.findWhere(list, properties)

rest `_.rest(array, [index])` *Alias:* ***tail***, ***drop***

Returns the **rest** of the elements in an array. Pass an **index** to return the values of the array from that index onward.

```
_.rest([5, 4, 3, 2, 1]);
=> [4, 3, 2, 1]
```

Εικόνα2.2.10 Λειτουργία της _.rest(array, [index])

isNaN `_.isNaN(object)`

Returns *true* if **object** is *NaN*.
Note: this is not the same as the native **isNaN** function, which will also return true if the variable is *undefined*.

```
_.isNaN(NaN);
=> true
isNaN(undefined);
=> true
_.isNaN(undefined);
=> false
```

Εικόνα2.2.11 Λειτουργία της _.isNaN(object)

isNumber `_.isNumber(object)`

Returns *true* if **object** is a Number (including `NaN`).

```
_.isNumber(8.4 * 5);
=> true
```

Εικόνα2.2.11 Λειτουργία της _.isNumber(object)

delay `_.delay(function, wait, [*arguments])`

Much like **setTimeout**, invokes **function** after **wait** milliseconds. If you pass the optional **arguments**, they will be forwarded on to the **function** when it is invoked.

```
var log = _.bind(console.log, console);
_.delay(log, 1000, 'logged later');
=> 'logged later' // Appears after one second.
```

Εικόνα2.2.12 Λειτουργία της _.delay(function, wait, [*arguments])

bind `_.bind(function, object, [*arguments])`

Bind a **function** to an **object**, meaning that whenever the function is called, the value of *this* will be the **object**. Optionally, pass **arguments** to the **function** to pre-fill them, also known as **partial application**.

```
var func = function(greeting){ return greeting + ': ' + this.name };
func = _.bind(func, {name : 'moe'}, 'hi');
func();
=> 'hi: moe'
```

Εικόνα2.2.13 Λειτουργία της _.bind(function, object, [*arguments])

sortBy `_.sortBy(list, iterator, [context])`

Returns a sorted copy of **list**, ranked in ascending order by the results of running each value through **iterator**. Iterator may also be the string name of the property to sort by (eg. `length`).

```
_.sortBy([1, 2, 3, 4, 5, 6], function(num){ return Math.sin(num); });
=> [5, 4, 6, 3, 1, 2]
```

Εικόνα2.2.14 Λειτουργία της _.sortBy(list, iterator, [contex])

groupBy `_.groupBy(list, iterator, [context])`

Splits a collection into sets, grouped by the result of running each value through **iterator**. If **iterator** is a string instead of a function, groups by the property named by **iterator** on each of the values.

```
_.groupBy([1.3, 2.1, 2.4], function(num){ return Math.floor(num); });
=> {1: [1.3], 2: [2.1, 2.4]}

_.groupBy(['one', 'two', 'three'], 'length');
=> {3: ["one", "two"], 5: ["three"]}
```

Εικόνα2.2.15 Λειτουργία της _.groupBy (list, iterator, [contex])

some `_.some(list, [iterator], [context])` *Alias:* ***any***

Returns *true* if any of the values in the **list** pass the **iterator** truth test. Short-circuits and stops traversing the list if a true element is found. Delegates to the native method **some**, if present.

```
_.some([null, 0, 'yes', false]);
=> true
```

Εικόνα2.2.16 Λειτουργία της _.some (list, [iterator], [contex])

bind `_.bind(function, object, [*arguments])`

Bind a **function** to an **object**, meaning that whenever the function is called, the value of *this* will be the **object**. Optionally, pass **arguments** to the **function** to pre-fill them, also known as **partial application**.

```
var func = function(greeting){ return greeting + ': ' + this.name };
func = _.bind(func, {name : 'moe'}, 'hi');
func();
=> 'hi: moe'
```

Εικόνα2.2.17 Λειτουργία της _.bind (function, [iterator], [contex])

compose `_.compose(*functions)`

Returns the composition of a list of **functions**, where each function consumes the return value of the function that follows. In math terms, composing the functions *f()*, *g()*, and *h()* produces *f(g(h()))*.

```
var greet    = function(name){ return "hi: " + name; };
var exclaim  = function(statement){ return statement + "!"; };
var welcome = _.compose(exclaim, greet);
welcome('moe');
=> 'hi: moe!'
```

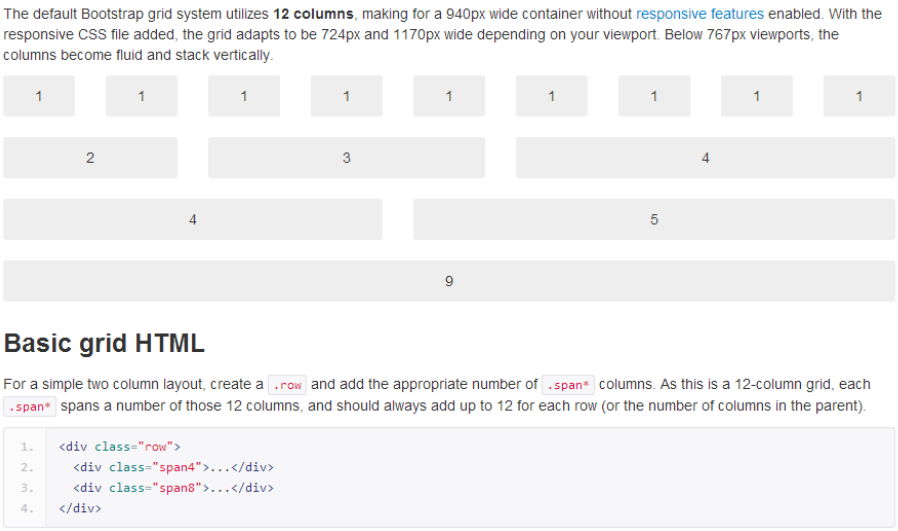
Εικόνα2.2.18 Λειτουργία της _.compose (*functions)

2.3 Bootstrap

Κομψό, έξυπνο και ισχυρό περιβάλλον χρήστη για ταχύτερη και ευκολότερη ανάπτυξη ιστοσελίδων. Το Bootstrap έρχεται εξοπλισμένο με HTML, CSS και JS για όλα τα είδη επιλογών. Το meteor υποστηρίζει τις βιβλιοθήκες του bootstrap και αυτό ήταν πολύ χρήσιμο για την ανάπτυξη του funny poker.

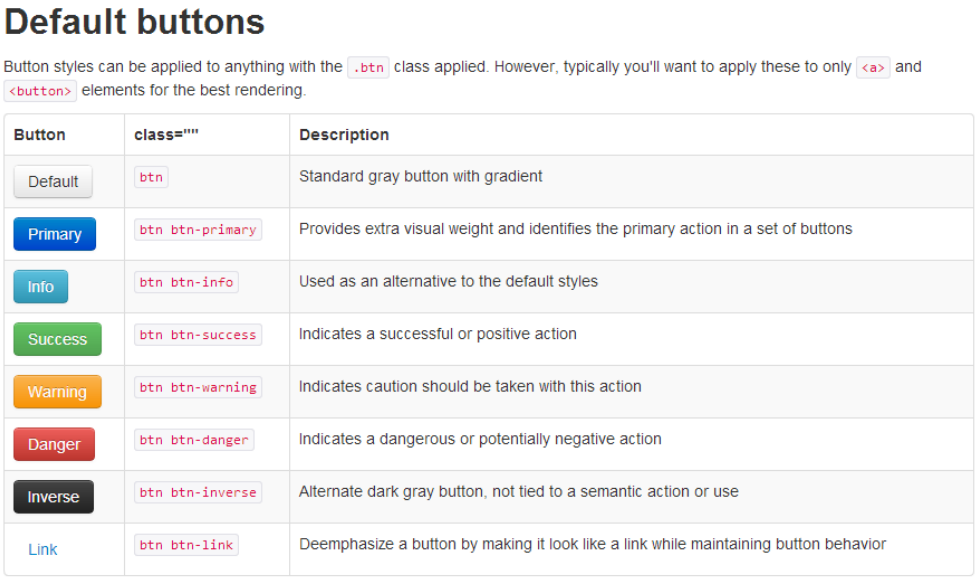
- Προσφέρει στις ενότητες docs:

Scaffolding: Παγκόσμιο στυλ για το σώμα της ιστοσελίδας για να επαναφέρετε τον τύπο και το φόντο, το στυλ σύνδεση, το δίκτυο κ.τ.λ. Συμφώνα με αυτό δημιουργήσαμε στο funny poker το σώμα της ιστοσελίδας.



Εικόνα2.3.1 Λειτουργία της row και της span .

Βάση CSS: Στυλ για κοινα HTML elements όπως η τυπογραφία, κωδικός, πίνακες, φόρμες, και τα κουμπιά. Στο Funny Poker το χρησιμοποιήσαμε για να φαίνεται με πράσινο χρώμα ο νικητής του παιχνιδιού.



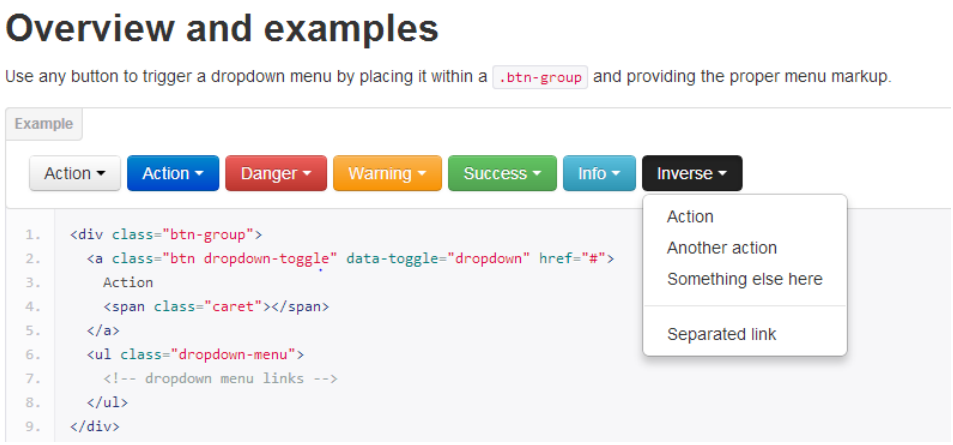
Default buttons

Button styles can be applied to anything with the `.btn` class applied. However, typically you'll want to apply these to only `<a>` and `<button>` elements for the best rendering.

Button	class=""	Description
Default	`btn`	Standard gray button with gradient
Primary	`btn btn-primary`	Provides extra visual weight and identifies the primary action in a set of buttons
Info	`btn btn-info`	Used as an alternative to the default styles
Success	`btn btn-success`	Indicates a successful or positive action
Warning	`btn btn-warning`	Indicates caution should be taken with this action
Danger	`btn btn-danger`	Indicates a dangerous or potentially negative action
Inverse	`btn btn-inverse`	Alternate dark gray button, not tied to a semantic action or use
Link	`btn btn-link`	Deemphasize a button by making it look like a link while maintaining button behavior

Εικόνα2.3.2 Λειτουργία της btn btn-success .

Εξαρτήματα: Βασικές μορφές των κοινών στοιχείων του περιβάλλοντος, όπως καρτέλες, ειδοποιήσεις, οι επικεφαλίδες των σελίδων, και πολλά άλλα. Στο Funny-poker χρησιμοποιήθηκε το dropdown menu όταν ο παίκτης έπρεπε να επιλέξει τι ποσό θα ποντάρει.



Overview and examples

Use any button to trigger a dropdown menu by placing it within a `.btn-group` and providing the proper menu markup.

Example

Action ▾ | Action ▾ | Danger ▾ | Warning ▾ | Success ▾ | Info ▾ | Inverse ▾

- Action
- Another action
- Something else here
- Separated link

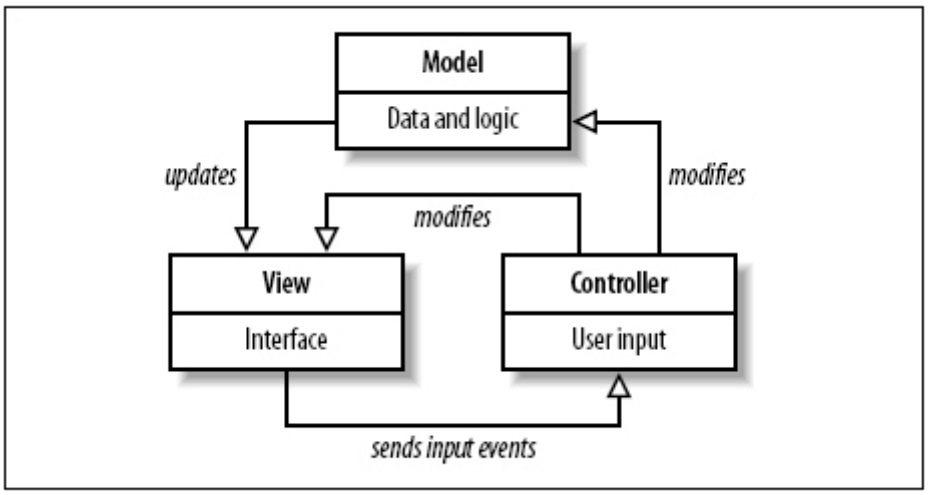
```
<div class="btn-group">
  <a class="btn dropdown-toggle" data-toggle="dropdown" href="#">
    Action
    <span class="caret"></span>
  </a>
  <ul class="dropdown-menu">
    <!-- dropdown menu links -->
  </ul>
</div>
```

Εικόνα2.3.3 Λειτουργία της btn dropdown-toggle

JavaScript plugins: Τα plugins JavaScript είναι διαδραστικά στοιχεία για τις επεξηγήσεις, popovers, modals, και πολλά άλλα.

2.4 Model–View–Controller

Το Meteor λειτουργεί σύμφωνα με το model-view-controller(MVC). Το MVC μπορεί να διαιρέσει ένα συστατικό σε τρία λογικά μέρη: μοντέλο (model), αναπαράσταση (view) και διαχείριση (controller) καθιστώντας ευκολότερη την διαδικασία τροποποίησης κάθε μέρους.



Εικόνα2.4 Λειτουργία του MVC.

Στην παραπάνω εικόνα συμβαίνουν τα εξής: Ο χρήστης δίνει κάποιο input στο site. Για παράδειγμα συμπληρώνει μία φόρμα και πατάει το κουμπί submit. Στη συνέχεια ο controller έχοντας λάβει το input του χρήστη μιλάει με το model χρησιμοποιώντας το input που του χρήστη σαν μεταβλητή και ζητάει δεδομένα από το model τα οποία όταν τα λαμβάνει τα προσαρμόζει ανάλογα με αυτό που ζήτησε ο χρήστης.Στη συνέχεια ο controller με τα ΝΕΑ δεδομένα πλέον , αλλάζει την view.

- **Model:** Στο model τοποθετούμε τις λειτουργίες της εφαρμογής που σχετίζονται με την πρόσβαση στη βάση δεδομένων. Οι λειτουργίες αυτές είναι με τη μορφή function(μεθόδων στον προγραμματισμό). Είναι κάποιες συναρτήσεις με τις οποίες εκτελούμε διάφορες λειτουργίες διαχείρισης των δεδομένων που λαμβάνουμε από τη βάση. Για παράδειγμα αν θέλουμε σε μία σελίδα να εμφανίσουμε κάποια βιβλία από τη βάση δεδομένων ,το πρώτο βήμα είναι ότι στο model των βιβλίων υπάρχει κάποια function για παράδειγμα "getAllBooks()" η οποία περιέχει κώδικα που μιλάει με τη βάση και τραβάει τα δεδομένα που θέλουμε.
- **View:** Μέσα στη view υπάρχει το HTML της σελίδας της εφαρμογής μας. Είναι αυτό που βλέπουμε. Τις περισσότερες φορές μία View μιλάει με ένα controller και αφού ο controller κάνει τις διάφορες επεξεργασίες των δεδομένων στέλνει στη View συγκεκριμένα δεδομένα να εμφανίσει.
- **Controller:** Ο controller είναι ο μεσίτης μεταξύ Model και της View. Ελέγχει το πώς "τρέχει" η εφαρμογή.Μιλάει με το Model, παίρνει τα δεδομένα που ζητά και εν συνεχεία και αφού τα επεξεργαστεί τα στέλνει πίσω στη View για απεικόνιση.Ας δούμε μία συνολική εικόνα για το MVC.

2.4.1 Πλεονεκτήματα του MVC:

Χτίζοντας μία εφαρμογή με MVC έχουμε τα εξής βασικά πλεονεκτήματα:

- **Διαχωρισμός Προβλημάτων (Separation of Concerns):**

Αυτό είναι και το πιο βασικό πλεονέκτημα του MVC. Ουσιαστικά δημιουργείται μία εφαρμογή η οποία έχει τρία επίπεδα, το επίπεδο των models , το επίπεδο των controllers και το επίπεδο των views και το κάθε επίπεδο επιτελεί ξεχωριστό έργο και ταυτόχρονα συνεργάζεται με τα άλλα επίπεδα. Μία σωστή MVC εφαρμογή είναι εκείνη που τα τρία επίπεδα είναι ξεκάθαρα καθορισμένα και δεν συμπλέκονται. Για παράδειγμα είναι λάθος στο επίπεδο των View να υπάρχει κώδικας που "μιλάει" με την βάση δεδομένων και "τραβάει" δεδομένα.

- **Επεκτασιμότητα:**

Το δεύτερο πλεονέκτημα της MVC αρχιτεκτονικής είναι επίσης πολύ σημαντικό. "Επεκτασιμότητα" είναι η δυνατότητα που διαθέτει μία εφαρμογή , κατά την οποία μπορούμε μελλοντικά να προσθέσουμε λειτουργίες σε αυτή ή να αλλάξουμε κάποιες από τις ήδη υπάρχουσες λειτουργίες και να έχουμε άλλα αποτελέσματα. Για να το δούμε εντελώς απλά και κατανοητά, η πλατφόρμα WordPress είναι επεκτάσιμη με τη χρήση των διάφορων plugins διότι προσθέτουμε στις ήδη υπάρχουσες λειτουργίες και άλλες λειτουργίες. Τα προγράμματα που είναι φτιαγμένα με MVC αρχιτεκτονική έχουν βασικό χαρακτηριστικό ότι είναι επεκτάσιμα.

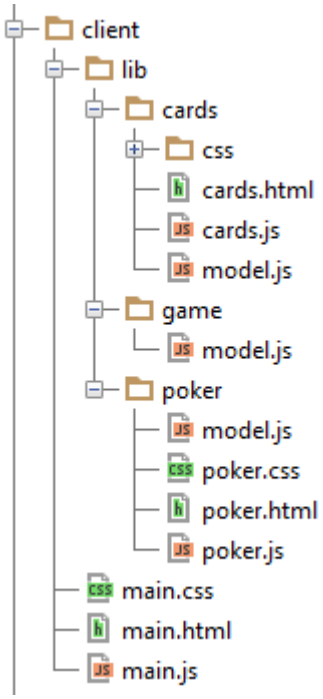
- **Ελεγξιμότητα (Testability):**

Αυτό είναι ένα πολύ κρίσιμο χαρακτηριστικό. Οι MVC εφαρμογές έχουν την δυνατότητα να είναι ελέγξιμες και με τον τρόπο αυτό συντηρούνται πιο εύκολα. Ας κάνουμε ένα απλό παράδειγμα. Έστω ότι έχουμε μία εφαρμογή η οποία διαθέτει μία λειτουργία login, δηλαδή ζητά από τον χρήστη να πληκτρολογήσει κάποια στοιχεία σε μία φόρμα και εν συνέχεια τον εισάγει μέσα στο σύστημα. Αυτή τη λειτουργία την ελέγχει κάποιος loginController ο οποίος περιέχει κώδικα που διαχειρίζεται τα δεδομένα αυτά που εισήχθησαν από τον χρήστη. Αυτός ο controller θεωρείται μία "μονάδα" ή αλλιώς unit. Στα MVC frameworks μπορούμε με πολλή ευκολία να γράψουμε απλό κώδικα-tests με τον οποίο τεστάρουμε αυτόν τον controller αλλά και κάθε μία από τις λειτουργίες του. Παίρνουμε τα αποτελέσματα και βλέπουμε η συγκεκριμένη μονάδα της εφαρμογής μας λειτουργεί σωστά.

- **"Καθαρά" URLs:**

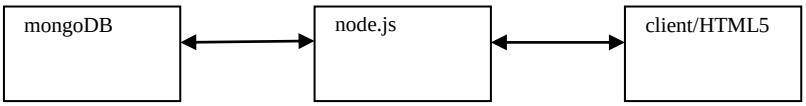
Τα περισσότερα MVC frameworks για web applications δίνουν τη δυνατότητα να έχουμε "καθαρά" urls. Με το MVC μπορούμε να έχουμε πιο όμορφα και εύληπτα από το χρήστη αλλά και την μηχανή αναζήτησης URLs.

Σε αυτή την πτυχιακή εργασία χρησιμοποιούμε το MVC μέσα στον client . Όπως βλέπουμε και την επόμενη εικόνα το αρχείο cards αποτελείται από το cards.html (αυτό που βλέπει ο χρήστης) , το model.js (το λειτουργικό κομμάτι των cards) και το cards.js (τον controller). Το ίδιο συμβαίνει και με το αρχείο poker . Ενώ για στο αρχείο game υπάρχει μόνο το λειτουργικό κομμάτι model.js αφού δεν το βλέπει καθόλου ο χρήστης. Στα επόμενα κεφάλαιο θα αναλύσουν όλα τα αρχεία του client.



Εικόνα2.4.1. Πως χρησιμοποιείται και για ποια αρχεία το MVC στο Funny Poker

ΚΕΦΑΛΑΙΟ 3: ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ



3.1 Βάση Δεδομένων mongoDB:

Το MongoDB είναι ένα αντικειμενοστραφές σύστημα NoSQL βάσεων δεδομένων που αναπτύχθηκε από την 10gen και είναι ένα λογισμικό ανοικτής αρχιτεκτονικής. Το όνομα MongoDB προέρχεται από τη λέξη «humongous». Η βάση δεδομένων προορίζεται να είναι κλιμακώσιμη και γρήγορη και είναι υλοποιημένη σε C++. Επιπλέον στην αντικειμενοστρέφεια του συστήματος αυτού, το MongoDB μπορεί να χρησιμοποιηθεί για την αποθήκευση και κατάτμηση μεγάλων δυαδικών αρχείων, όπως εικόνων και βίντεο. Το σύστημα είναι σταθερό και ανεκτικό στα λάθη, ενώ παρέχει μία πολύπλοκη γλώσσα ερωτημάτων όπως και μία υλοποίηση του MapReduce.

3.1.1 Το μοντέλο δεδομένων του MongoDB

Το MongoDB αποθηκεύει έγγραφα με τη μορφή των BSON αντικειμένων, που είναι δυαδικά κωδικοποιημένα JSON αντικείμενα. Η μορφοποίηση BSON υποστηρίζει εμφωλευμένες δομές αντικειμένων με ενσωματωμένα αντικείμενα και πίνακες, όπως κάνει και το JSON. Το MongoDB υποστηρίζει αλλαγές στις ιδιότητες, άρα αν μία μόνο ιδιότητα αλλάξει στην εφαρμογή, τότε μόνο αυτή στέλνεται πίσω στη βάση δεδομένων. Κάθε έγγραφο έχει ένα πεδίο ID, που χρησιμοποιείται ως το πρωτεύων κλειδί. Για την υποστήριξη γρήγορων ερωτημάτων, ο σχεδιαστής μπορεί να δημιουργήσει ένα ευρετήριο για ένα πεδίο ενός εγγράφου που θα δεχθεί ερωτήσεις. Το MongoDB υποστηρίζει επίσης, την ευρετηριοποίηση ενσωματωμένων αντικειμένων και πινάκων.

Το MongoDB έχει ένα ειδικό γνώρισμα για τους πίνακες που λέγεται «πολλαπλά κλειδιά». Το γνώρισμα αυτό επιτρέπει τη χρησιμοποίηση ενός πίνακα ως ευρετήριο, που μπορεί για παράδειγμα να περιέχει λέξεις κλειδιά για ένα έγγραφο. Με ένα τέτοιο ευρετήριο, τα έγγραφα μπορούν να αναζητούνται σύμφωνα με τις λέξεις κλειδιά. Τα έγγραφα στο MongoDB μπορούν να οργανωθούν σε συλλογές. Κάθε συλλογή μπορεί να περιέχει οποιοδήποτε τύπο εγγράφων, αλλά τα ερωτήματα και τα ευρετήρια μπορούν να γίνουν μόνο μέσα στις συλλογές.

Επειδή το MongoDB έχε ένα περιορισμό 40 ευρετηρίων ανά συλλογή, είναι προτιμότερη η χρησιμοποίηση μίας συλλογής για κάθε τύπο εγγράφου, ούτως ώστε να έχουν τα ερωτήματα καλύτερη απόδοση. Οι σχέσεις στο MongoDB μπορεί να απεικονιστεί μέσω ενσωματωμένων αντικειμένων και πινάκων. Οπότε το μοντέλο δεδομένων πρέπει να έχει τη μορφή δένδρου. Η πρώτη επιλογή υπονοεί ότι μερικά έγγραφα πρέπει να υπάρχουν σε διπλότυπα μέσα στη βάση δεδομένων. Αυτή η λύση πρέπει να χρησιμοποιείται μόνο αν τα διπλότυπα έγγραφα δε χρειάζονται συχνές ανανεώσεις. Η δεύτερη επιλογή είναι η χρησιμοποίηση συζεύξεων στην πλευρά των πελατών που μπορούν να τοποθετηθούν σε μορφή δένδρου. Αυτό απαιτεί περισσότερη δουλειά στο επίπεδο εφαρμογής και αυξάνει την διαδικτυακή επικοινωνία με τη βάση δεδομένων.

3.1.2 Η αρχιτεκτονική του MongoDB

Ένα cluster του MongoDB απαρτίζεται από τρία συστατικά:

- **Κόμβοι Shard**

Οι κόμβοι shard είναι υπεύθυνοι για την αποθήκευση των πραγματικών δεδομένων. Κάθε τέτοιος κόμβος μπορεί να απαρτίζεται είτε από έναν κόμβο είτε από ένα ζευγάρι κόμβων που έχουν την ίδια πληροφορία. Σε μελλοντικές εκδόσεις του MongoDB, ένας κόμβος shard μπορεί να απαρτίζεται από περισσότερους από δύο κόμβους για καλύτερη απόδοση.

- **Εξυπηρετητές παραμετροποίησης**

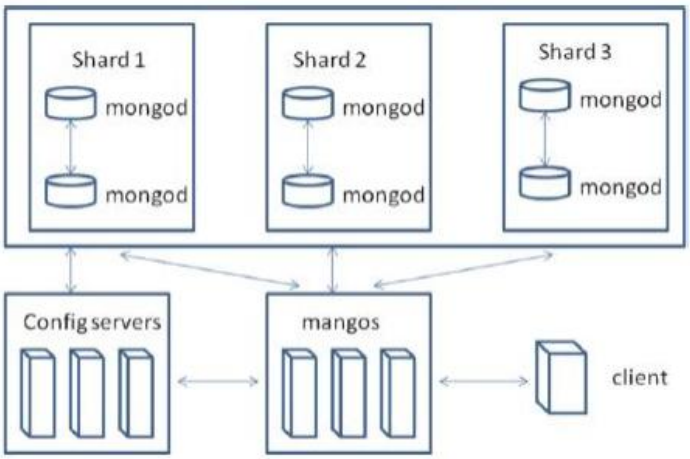
Οι εξυπηρετητές παραμετροποίησης χρησιμοποιούνται για να αποθηκεύουν τις μεταπληροφορίες και την πληροφορία δρομολόγησης του cluster του MongoDB και είναι προσβάσιμα μέσω των κόμβων shard και των υπηρεσιών δρομολόγησης.

- **Υπηρεσίες δρομολόγησης ή mongos**

Τα mongos είναι υπεύθυνα για την απόδοση των διεργασιών που αιτούνται οι χρήστες. Ανάλογα με τον τύπο των λειτουργιών, τα mongos στέλνουν αιτήσεις στους απαραίτητους κόμβους shard και συνενώνουν τα αποτελέσματα πριν αυτά επιστρέψουν στον πελάτη. Τα mongos μπορούν να εκτελούνται παράλληλα.

Το MongoDB είναι υλοποιημένο με τη γλώσσα προγραμματισμού C++ και αποτελείται από δύο τύπους υπηρεσιών: τον πυρήνα της βάσης δεδομένων που καλείται mongod και τις υπηρεσίες mongos. Το MongoDB για την αποθήκευση χρησιμοποιεί αρχεία που έχουν χαρτογραφηθεί στη μνήμη, και αφήνει στον διαχειριστή εικονικής μνήμης του λειτουργικού συστήματος να αποφασίσει ποια μέρη της βάσης δεδομένων θα αποθηκευτούν στη μνήμη και ποια θα μείνουν στον σκληρό δίσκο.

Τα έγγραφα στο MongoDB cluster τμηματοποιούνται από τη δική τους συλλογή και από τις επιθυμίες του χρήστη, που καλείται κλειδί shard. Το κλειδί shard είναι παρόμοιο με ένα ευρετήριο και διαθέτει πολλαπλά πεδία.



Εικόνα3.1.2 Αρχιτεκτονική του MongoDB.

3.2 ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ FUNNY POKER:

Τώρα που αναλύσαμε την λειτουργικότητα του mongoDB θεωρητικά ας το δούμε πως λειτουργεί στην συγκεκριμένη εφαρμογή. Πρώτα από όλα χρησιμοποιούμε δύο βάσεις δεδομένων την dbusers για την εγγραφή των παικτών στην σελίδα και την dbwallet όπου αποθηκεύονται τα εικονικά χρήματα του κάθε χρήστη.

```
meteor  0.03125GB
> show collections
system.indexes
users
wallets
>
```

Εικόνα3.2: collections της MongoDB

3.2.1 users database

Η meteor προσφέρει το meteor account system, αυτό το σύστημα είναι ένα πακέτο λογαριασμών βάσης που συνήθως συνδέονται με λογαριασμούς facebook, google και twiter. Έτσι , προσθέτοντας στον client την εντολή:

```
Accounts.ui.config({

   passwordSignupFields: 'USERNAME_AND_OPTIONAL_EMAIL'

});
```

Αυτόματα η meteor δημιουργεί την mongo βάση που κρατάει τα στοιχεία των users.

Accounts.ui.config(options) *Client*

Configure the behavior of `{{loginButtons}}`.

Options

requestPermissions Object

Which permissions to request from the user for each external service.

requestOfflineToken Object

To ask the user for permission to act on their behalf when offline, map the relevant external service to `true`. Currently only supported with Google. See Meteor.loginWithExternalService for more details.

passwordSignupFields String

Which fields to display in the user creation form. One of 'USERNAME_AND_EMAIL', 'USERNAME_AND_OPTIONAL_EMAIL', 'USERNAME_ONLY', or 'EMAIL_ONLY' (default).

Example:

```
Accounts.ui.config({
  requestPermissions: {
    facebook: ['user_likes'],
    github: ['user', 'repo']
  },
  requestOfflineToken: {
    google: true
  },
  passwordSignupFields: 'USERNAME_AND_OPTIONAL_EMAIL'
});
```

Εικόνα 3.2.1 : Λειτουργικότητα της Accounts.ui.config(options)

Στην επόμενη εικόνα φαίνονται τα πεδία της users βάσης δεδομένων. Βλέπουμε ότι σε κάθε χρήστη αντιστοιχεί ένα id, το email , το password και φυσικά το username.

Εικόνα 3.2.2 db.users.find()

3.2.2 wallets database

Για να δημιουργήσουμε την βάση δεδομένων με όνομα wallet αρκεί να τρέξουμε στον server την επόμενη εντολή:

Wallets = new Meteor.Collection("wallets");

Η βάση δεδομένων wallet κρατάει τα χρήματα του παίκτη. Ο χρήστης έχει την δυνατότητα να μεταφέρει χρήματα από έναν εικονικό λογαριασμό στη εφαρμογή να τα παίξει και αν κερδίσει το παιχνίδι του προστίθενται τα κέρδη στο λογαριασμό του και αντίστοιχα του αφαιρείται το ποσό πού έπαιξε σε περίπτωση ήττας.

Η Walletsdb συνδέεται με την usersdb με το πεδίο username. Επιπλέον διαθέτει πεδίο με όνομα amount όπου εκχωρείται το χρηματικό ποσό που έχει την εκάστοτε στιγμή ο παίχτης κάθως και ένα id.

```
> db.wallets.find()
{ "_id" : "8qcDZQMQRK5TeB6QJ", "amount" : 10880, "username" : "antwnia" }
{ "username" : "antwnia", "amount" : 300, "_id" : "yDkCWdYZByj6pQP3Q" }
{ "username" : "anta", "amount" : 300, "_id" : "wzik2NKybAL5JWbLB" }
{ "username" : "kwstas", "amount" : 560, "_id" : "GB2hni8WPxjSgzF5m" }
{ "username" : "giwrgos", "amount" : 820, "_id" : "tZMR56jr9DCeD55hE" }
>
```

Εικόνα3.2.2.1 db.wallets.find()

Τρέχοντας στον server την παρακάτω εντολή πρώτον συνδέουμε το username από την usersdb με την walletsdb με τα οπου δημιουργείται η εκάστοτε λογαριασμός. Παράλληλα , η εφαρμογή παραχωρεί σε κάθε καινούργιο παίκτη 300 ευρώ στο πεδίο amount.

```
Meteor.startup(function () {
    Accounts.onCreateUser(function(options, user) {

        Wallets.insert({username: user.username, amount: 300});

        // We still want the default hook's 'profile' behavior.
        if (options.profile) {
            user.profile = options.profile;
        }
        return user;
    });
});
```

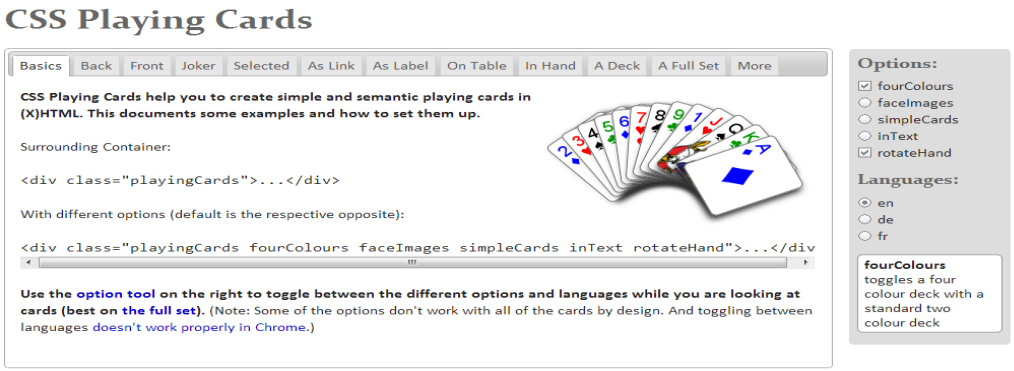
Εικόνα3.2.2.2 Συμπεριφορά της walletsdb.

ΚΕΦΑΛΑΙΟ 4 : ΑΝΑΛΥΣΗ CARDS

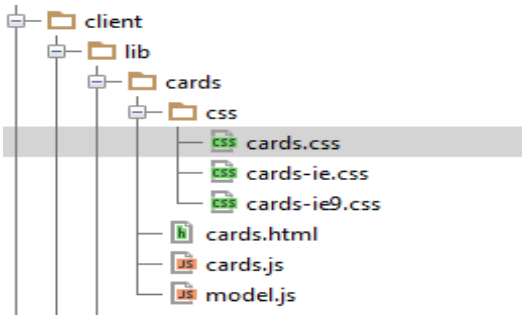
Το αρχείο cards αποτελείται από τα cards.html, cards.js, model.js. Όλα μαζί είναι υπεύθυνα για την εμφάνιση των χαρτιών στην ιστοσελίδα, καθώς και για την αλληλεπίδρασή των φύλλων με τους παίκτες.

4.1 Γραφικά της τράπουλας

Στην εφαρμογή χρησιμοποιείται τράπουλα των 52 φύλλων. Το γραφικό κομμάτι των τραπουλόχαρτων είναι ένα css αρχείο που δημιούργησε η κ. Anika Henke και υπάρχει στην http://selfthinker.github.io/CSS-Playing-Cards/ .



Εικόνα4.1.1 cards.css



Εικόνα4.1.2 Το css αρχείο τρέχει μέσα στον φάκελο cards του client στο meteor .

4.2Αρχεία του cards

4.2.1 Αρχείο cards.html

Το αρχείο αυτό είναι αυτό που βλέπει ο χρήστης. Το html αρχείο αποτελείται από δύο templates την cardHolder και την card.Ο cardholder είναι οποιοσδήποτε κρατάει φύλλα για την εφαρμογή αυτή είναι ο πραγματικός παίκτης, ο υπολογιστής-αντίπαλος, η τράπουλα (deck) και ο κάδος (bin) που πάνε τα αλλαγμένα φύλλα. Cards όπως υποδηλώνει και το όνομα είναι τα φύλλα της τράπουλας, τα οποία έχουν δύο όψεις την μπροστά και την πίσω. Επίσης οι κάρτες έχουν και δύο ιδιότητες τον αριθμό (rank) και τον χρώμα (suit). Την συμπεριφορά των templates την ορίζει το αρχείο model.js που θα αναλύσουμε αργότερα. Ενώ το αρχείο card.js ενώνει model.js με το αρχείο card.html.

```
<template name="cardholder">
    <div class="playingCards">
        <ul class="table">
            {{#each cards}}
            <li>
                {{>card}}
            </li>
            {{/each}}
        </ul>
    </div>
</template>

<template name="card">
    {{#if strong}}
    <strong>
    {{/if}}
    {{#if isBack}}
        <div class="card back">*</div>
    {{else}}
        <div class="card {{rankClass}} {{suitClass}}">
            <span class="rank">{{rank}}</span>
            <span class="suit">&{{suit}};</span>
        </div>
    {{/if}}
    {{#if strong}}
    </strong>
    {{/if}}
</template>
```

4.2.2 Αρχείο model.js του card

Όπως ήδη έχουμε αναφέρει στο model τοποθετούμε τις λειτουργίες της εφαρμογής που σχετίζονται με την πρόσβαση στη βάση δεδομένων. Οι λειτουργίες αυτές είναι με τη μορφή function (μεθόδων στον προγραμματισμό). Είναι συναρτήσεις με τις οποίες εκτελούμε λειτουργίες διαχείρισης των δεδομένων που λαμβάνουμε από τη βάση.

Το αντικείμενο card περιέχει:

- Card.Ranks : ένα array από τους αριθμούς (ranks) της τράπουλας.
- Card.Suites : ένα array από τα χρώματα (suits) της τράπουλας.
- Card.Holder : κλάση του cardholder αντικειμένου.

```
Card = {
    Ranks : [
        {value:2,   rankClass:"rank-2",  weight: 13},
        {value:3,   rankClass:"rank-3",  weight: 12},
        {value:4,   rankClass:"rank-4",  weight: 11},
        {value:5,   rankClass:"rank-5",  weight: 10},
        {value:6,   rankClass:"rank-6",  weight: 9},
        {value:7,   rankClass:"rank-7",  weight: 8},
        {value:8,   rankClass:"rank-8",  weight: 7},
        {value:9,   rankClass:"rank-9",  weight: 6},
        {value:10,  rankClass:"rank-10", weight: 5},
        {value:"J", rankClass:"rank-j",  weight: 4},
        {value:"Q", rankClass:"rank-q",  weight: 3},
        {value:"K", rankClass:"rank-k",  weight: 2},
        {value:"A", rankClass:"rank-a",  weight: 1}
    ],
    Suits : [
        "hearts","clubs","diams","spades"
    ]
};
```

Εικόνα4.2.2.1 Δήλωση των ranks και suits στην card.

Κάθε κάρτα του this.cards έχει τα ακόλουθα χαρακτηριστικά:

- **CardHolderName**: το όνομα του κατόχου της κάρτας σε μια συγκεκριμένη χρονική στιγμή.
- **Order**: η σειρά αυτής της κάρτας στη συλλογή των καρτών του κατόχου της κάρτας.
- **Suit**: το χρώμα της κάρτας. Μία από τις τιμές της Card.Suits.
- **Rank**: η κατάταξη αυτής της κάρτας. Μία από τις τιμές της Card.Rank.
- **IsPicked**: μια boolean ένδειξη που δείχνει αν μια κάρτα έχει επιλεχτεί σε μια συγκεκριμένη φάση του παιχνιδιού.
- **IsTurned**: μια boolean ένδειξη που δείχνει αν αυτή η κάρτα είναι κρυμμένη δηλαδή γυρισμένη από την ανάποδα πλευρά στο τραπέζι, ώστε να είναι ορατή μόνο στον κάτοχο της κάρτας .
- **RankClass**: η rank CSS που πρόκειται να χρησιμοποιηθεί κατά την τρέχουσα χρονική στιγμή.
- **SuitClass**: το suit CSS που πρόκειται να χρησιμοποιηθεί κατά την τρέχουσα χρονική στιγμή.

Η Card.Holder είναι μια function που περιέχει το CardholderName (το string όνομα του παίκτη) , το deck (την τράπουλα), το bin (το σημείο που πηγαίνουν τα φύλλα που θέλει ο παίκτης να αλλάξει για να τα αντικαταστήσει με καινούργια), το cards (τα φύλλα της τράπουλας) και τέλος το newDeck (τα υπόλοιπα φύλλα της τράπουλας μετά το μοίρασμα).

```
Card.Holder = function(cardholderName, deck, bin, cards, newDeck) {
    if (_.isUndefined(cardholderName) || !_.isString(cardholderName)) {
        throw new Error("Card.Holder.constructor ---> The creation of a nameless Card.holder is not allowed");
    }

    var self = this;
    self.name = cardholderName;
    self.deck = _.isUndefined(deck) ? null : deck;
    self.bin = _.isUndefined(bin) ? null : bin;
    self.cards = new Meteor.Collection(null);
```

Η μεταβλητή self περιέχει :

1.)name: Το string όνομα του cardholder.
2.)deck : Τράπουλα που είναι null αν δεν βρεθεί.
3.)bin: Το σημείο στο οποίο πηγαίνουν τα φύλλα που άλλαξαν οι παίκτες.
4.)cards: Βάση δεδομένων που τρέχει στον browser που περιέχει τις κάρτες του αντικειμένου cardHolder.

Η δημιουργία τράπουλας γίνεται ως εξής σε καθένα από τα Card.Rank δημιουργούμε μια μέθοδο με όρισμα rank αντίστοιχα για καθένα από τα Card.Suit δημιουργούμε μια μέθοδο με όρισμα suit .Έπειτα καταχωρούμε στην self.cards το όνομα του card.Holder, order, suit, rank, rankWeight, isPicked, isTurned, rankClass και suitClass.

```
    if (newDeck) {
        var numCards = 0;
        _.each(Card.Ranks,
            function (rank) {
                _.each(Card.Suits,
                    function (suit) {
                        self.cards.insert({
                            cardholderName: self.name,
                            order: numCards++,
                            suit:suit,
                            rank:rank.value,
                            rankWeight: rank.weight,

                            isPicked: false,
                            isTurned: false,

                            rankClass: rank.rankClass,
                            suitClass:suit
                        });
                    }
                );
            }
        );
    } else if (cards) {
        self.add(cards);
    }

    Card.Holder._put(cardholderName, self);
};
```

Prototype members του Card.Holder:

- getCards(): επιστρέφει ένα JS array από την cards collection που έχει ο cardholder.

```
Card.Holder.prototype.getCards = function (whereConditions, actions) {
    var _actions = _.isUndefined(actions)? {} : actions;
    var _whereConditions = _.isUndefined(whereConditions)? {} : whereConditions;

    return this.cards.find(_whereConditions, _actions).fetch();
};
```

- add(cards): προσθέτει ένα array καρτών μέσα στην cards collection αυτού του cardholder.

```
Card.Holder.prototype.add = function (cards) {
    var self = this;

    if (_.isUndefined(cards)  || !_.isArray(cards)) {
        return;
    }
    _.each(cards, function(card) {
        card.cardholderName = self.name;
        self.cards.insert(card);
    });
};
```

- shuffle(): ανακάτεμα των καρτών από την βάση δεδομένων αλλάζοντας τις τιμές στην σειρά των καρτών.

```
Card.Holder.prototype.shuffle =  function() {
    var self = this;
    var cards = _.shuffle(self.getCards());
    self.cards.remove({});
    self.add(cards);
```

- replace(card, from, to): αντικατάσταση της κάρτας "card" του ενός cardholder με την πρώτη κάρτα του cardholder "from" και αυτή την κάρτα την προσθέτει στον cardholder "to".

```
Card.Holder.prototype.replace = function (card, from, to) {
    var self = this;
    if (!self.cards.findOne(card) || ! from.cards || ! to.cards) {
        return null;
    }

    console.log("In CreateCardHolder("+self.name+").replace --> replacing "+card.suit+" "+card.rank);

    var newCard = from.cards.findOne();
    from.cards.remove(newCard);
    self.cards.remove(card);

    newCard.cardholderName = self.name;
    self.cards.insert(newCard);

    card.cardholderName = to.name;
    to.cards.insert(card);

    console.log("In CreateCardHolder("+self.name+").replace --> replaced with "+newCard.suit+" "+newCard.rank);

    return newCard;
};
```

- pass(from, numOfCards): διαγράφει την πρώτη από την "numOfCards" κάρτα του cardholder "from" και την προσθέτει μέσα σε αυτόν τον cardholder.

```
Card.Holder.prototype.pass = function(cardHolder, numOfCards) {
    var self = this;
    var cards = _.first(self.getCards(), _.isUndefined(numOfCards) ? 1 : numOfCards);
    _.each(cards, function (card) { self.cards.remove(card); });
    cardHolder.add(cards);
};
```

- hide(): σημαδεύει τις κρυμμένες κάρτες του cardholder όταν γίνεται αληθής "isTurned" κάποιας ή όλων των καρτών του cardholder.

```
Card.Holder.prototype.hide = function() {
    this.cards.update({}, {$set: {isTurned: true}}, { multi: true });
};
```

- log(): εμφανίζει στην κονσόλα τις κάρτες του cardholder.

```
Card.Holder.prototype.log = function () {
    var self = this;
    _.each(self.getCards(), function(card) {
        console.log(_.values(card).toString());
    });
};
```

- Static members του Card.Holder:

- get(cardholderName)
- put(cardholderName, cardolder)

4.2.3 Αρχείο card.js

Το card.js στην ουσία δένει το model.js με το card.html. Το card.html είναι αυτό που βλέπει ο χρήστης , το αρχείο model.js είναι αυτό που αντιλαμβάνεται η μηχανή , το λειτουργικό κομμάτι των card, και τέλος, το αρχείο card.js ενώνει το εικονικό με το λειτουργικό κομμάτι των cards. Εδώ δημιουργούμε τις εξής τέσσερις μεθόδους:

- Την cards που επιστρέφει τις κάρτες του cardholder στην σειρά.

```
Template.cardholder.cards = function () {
    return this.getCards({}, {sort: {order : -1}});
};
```

- Την strong , πρόκειται για μια Boolean μεταβλητή που είναι αληθής όταν ο παίκτης επιλέξει το φύλλο.

```
Template.card.strong = function () {
    return this.isPicked;
};
```

- Την isBack , πρόκειται για μια Boolean μεταβλητή που είναι αληθής όταν το φύλλο είναι από την ανάποδη πλευρά.

```
Template.card.isBack = function () {
    return this.isTurned;
};
```

- Τα events που περιέχουν

-click: Με click στην κάρτα επιλέγει ο παίκτης την κάρτα που θέλει να αλλάξει ,αν της ξανά κάνει click τότε σταματά να είναι επιλεγμένη.

- double click: Με διπλό click πάνω στην κάρτα το φύλλο γυρνάει από την πίσω πλευρά-κρύβεται. Με δεύτερο διπλό click πάνω στην κάρτα, γυρνάει πάλι από τη κανονική πλευρά του φύλλου.

```
Template.card.events({
    'dblclick': function () {
        //Card.Holder.get(this.cardholderName).turn(this);
        Card.Holder.get(this.cardholderName).cards.update({_id:this._id}, {$set: {isTurned: !this.isTurned}});
    },
    'click': function () {
        //Card.Holder.get(this.cardholderName).pick(this);
        Card.Holder.get(this.cardholderName).cards.update({_id:this._id}, {$set: {isPicked: !this.isPicked}});
    }
});
```

ΚΕΦΑΛΑΙΟ 5 : ΑΝΑΛΥΣΗ GAME

Μέσα στο game υπάρχουν οι μέθοδοι που είναι αρμόδιοι για την λειτουργία της συγκεκριμένης παραλλαγής του poker. Αποτελείται από την model.js που θα αναλυθεί ακριβώς από κάτω.

5.1 model.js του game

Ακολουθεί ανάλυση του κώδικα που τρέχει μεσα στο model.js του game.

- Games.Phases (phases) : Η μέθοδος Game.Phases μοντελοποιεί τις φάσεις του παιχνιδιού. Η self.phases επιστρέφει ένα array με order ,name (το όνομα της φάσης) , ένα array με τους παίκτες .Ενώ παράλληλα με την _.bind ,που αναλύθηκε σε προηγούμενο κεφάλαιο, δεσμεύει σε αυτήν την self στις functions phase.onStart, phase.onEnd και phase.onPlayerAdvance

```
Game.Phases = function (phases) {
    var self = this;
    self.phases = [];
    self.currentPhaseIndex = 0;

    _.each(phases,
        function(phase) {
            self.phases.push({
                order: self.phases.length,
                name: phase.name,
                players: [],
                onStart: _.bind(phase.onStart, self),
                onEnd: _.bind(phase.onEnd, self),
                onPlayerAdvance: _.bind(phase.onPlayerAdvance, self)
            });
        });
};
```

Χρησιμοποιώντας την _.extend προσθέτουμε properties στην μεταβλητή phases. Οι μέθοδοι που δημιουργούμε είναι οι εξής :

- getCurrentPhase() : Αναγνωρίζει σε ποια φάση βρίσκεται ο παίκτης από το ευρετήριο currentIndex του this.

```
getCurrentPhase: function() {
    return this.phases[this.currentPhaseIndex];
},
```

- getNextPhase(): Επιστρέφει την επόμενη φάση του παιχνιδιού από το ευρετήριο των φάσεων.

```
*/
getNextPhase: function() {
    return this.currentPhaseIndex < this.phases.length?
        this.phases[this.currentPhaseIndex+1] : null;
},
/**
```

- getPhase(): Κρατάει το όνομα της φάσης στην οποία βρίσκεται ο παίκτης.

```
getPhase: function(phaseName) {
    return _.isUndefined(phaseName) ? null : _.findWhere(self.phases, {name: phaseName});
},
/**
```

- isCurrentPhase(phase name) : Δηλώνει την παράμετρο phaseName από την name της getCurrentPhase().

```
*/
isCurrentPhase: function(phaseName) {
    return this.getCurrentPhase().name == phaseName;
},
/**
```

- getCurrentPlayer(): Βρίσκει τον πρώτο από τους παίκτες που βρίσκεται σε κατάσταση φάσης παιχνιδιού στην getCurrentPhase() της this.

```
getCurrentPlayer: function () {
    return _.first(this.getCurrentPhase().players);
},
```

- isCurrentPlayer(player) : Δήλωση παραμέτρου player ως τον πρώτο από τους παίκτες που βρίσκεται στην getCurrentPhase().

```
isCurrentPlayer: function (player) {
    return _.first(this.getCurrentPhase().players) == player;
},
```

- advanceCurrentPlayer(phaseName): Μέθοδος (function) με γνώρισμα την PhaseName. Εδώ δηλώνεται ότι αν ο πρώτος από παίκτες (isHuman) πάει στην επόμενη φάση τότε ο παίκτης που απέμεινε (isEngine) θα παραμείνει στην προηγούμενη φάση. Αυτό συμβαίνει για λειτουργικούς λόγους πρέπει πρώτα να ποντάρει ο πραγματικός παίκτης και μετά να δεχτεί η μηχανή. Σε αυτή την μέθοδο ,επίσης, δηλώνεται ότι όταν είναι το παιχνίδι onEnd τότε είναι και onStart.

```
advanceCurrentPlayer: function(phaseName) {
    var self = this;
    var thisPhase = self.getCurrentPhase();
    var nextPhase = self.getPhase(phaseName) || self.getNextPhase();

    if (nextPhase) {
        nextPhase.players.push(_.first(thisPhase.players));
        thisPhase.players = _.rest(thisPhase.players);
    }
    if (thisPhase.players.length == 0) {
        thisPhase.onEnd();
        if (self.currentPhaseIndex < self.phases.length) {
            self.currentPhaseIndex++;
            nextPhase.onStart();
        }
    }
    self.getCurrentPhase().onPlayerAdvance();
},
/**
```

- startGame() : Όταν η phases της self είναι 0 το παιχνίδι βρίσκεται είτε στην φάση onStart είτε στην φάση onEnd. Στο επόμενο κεφάλαιο αναλύεται η phases της self. Αν υπάρχει παίκτης στην φάση 0 του παιχνιδιού τον πάει αμέσως στην επόμενη φάση και το currentPhaseIndex γίνεται 1.

```
startGame: function() {
    var self = this;
    self.phases[0].onStart();
    self.phases[0].onEnd();
    if (self.phases[0].players.length == 0) {
        self.currentPhaseIndex = self.phases.length - 1;
    } else {
        self.phases[1].players = self.phases[0].players;
        self.phases[0].players = [];
        self.currentPhaseIndex = 1;
    }
    self.getCurrentPhase().onStart();
}
```

ΚΕΦΑΛΑΙΟ 6 : ΑΝΑΛΥΣΗ POKER

Ο κύκλος ζωής ενός παιχνιδιού πόκερ (οι φάσεις ενός παιχνιδιού πόκερ) αποτελείται από τις ακόλουθες φάσεις: "Start", "BetFirstTime", "ChangeCards", "BetSecondTime", "τέλος". Όλοι οι παίκτες ξεκινούν στην "έναρξη" φάση και προχωρούν ένα προς ένα με στην δεύτερη φάση και στη συνέχεια στη τρίτη κλπ. Αυτή η κατηγορία είναι κληρονομείται από την κλάση Game.Phases που βρίσκεται στο model.js του game που αναλύσαμε ακριβώς από πάνω. Επίσης εδώ δηλώνονται και οι συνδυασμοί που μπορεί να επιτύχει κάθε παίκτης και ο τρόπος που αναγνωρίζεται ο νικητήριος συνδυασμός. Τέλος στο model.js αυτού του αρχείου φαίνεται και ο τρόπος που επιλέγει ποια φύλλα θα κρατήσει ο παίκτης-μηχανή.

Όπως το αρχείο cards έτσι και το poker λειτουργεί σύμφωνα με το MVC που περιγράψαμε πιο πάνω και περιέχει το poker.html, το model.js και το poker.js.

6.1 Αρχείο poker.html

Στην δημιουργία αυτού του αρχείο χρησιμοποιήθηκαν τα components.html της boostrap. Το bootstrap , όπως ήδη αναφέρθηκε, είναι μια βιβλιοθήκη με εντολές σε html , javascript και css για εύκολη και γρήγορη ανάπτυξη ιστοσελίδων. Μέσω αυτού δημιουργούμε τα κουμπιά (buttons) καθώς και το μενού από το οποίο επιλέγει ο χρήστης τι ποσό θα ποντάρει. Ειδικότερα, το όνομα του νικητή του παιχνιδιού στα αριστερά «πρασινίζει». Ο παίκτης isHuman έχει την δυνατότητα να ποντάρει (bet) και να αλλάξει φύλλα (Change Cards).

```
<template name="pokerPlayer">
<div class="row-fluid">
    <div class="span-2" >
        {{#if isWinner}}
        <span class="muted btn-large btn-success" style="...">{{playerName}}</span>
        {{else}}
        <h4 class="muted" style="...">
            {{playerName}}
        </h4>
        {{/if}}
    </div>
    <div class="offset2">
        {{>cardholder}}
    </div>
    <div class="offset9">
        <div class="btn-group-vertical">
            {{#if isHuman}}
            <button class="btn btn-info"> {{currentBet}} Euro</button>
            {{/if}}

            {{#if canChangeCards}}
            <button class="btn btn-success poker-change-cards" >Change Cards</button>
            {{/if}}
```

```
        {{#if canChangeCards}}
        <button class="btn btn-success poker-change-cards" >Change Cards</button>
        {{/if}}

        <div class="btn-group poker-bet">
            {{#if canBet}}
            <a class="btn btn-primary" href="#"><i class="icon-thumbs-up icon-white"></i> Bet</a>
            <a class="btn btn-primary dropdown-toggle" data-toggle="dropdown" href="#"><span class="caret"></span></a>
            <ul class="dropdown-menu">
                <li class="poker-bet-0"><a href="#"><i  class="icon-thumbs-up"></i>     0 euro</a></li>
                <li class="poker-bet-10"><a href="#"><i class="icon-thumbs-up"></i>   10 euro</a></li>
                <li class="poker-bet-20"><a href="#"><i class="icon-thumbs-up"></i>   20 euro</a></li>
                <li class="poker-bet-50"><a href="#"><i class="icon-thumbs-up"></i>   50 euro</a></li>
                <li class="divider"></li>
                <li class="poker-bet-200"><a href="#"><i class="icon-thumbs-up"></i>  200 euro</a></li>
            </ul>
            {{/if}}
        </div>
    </div>

    </div>
</div>
</template>
```

6.2 Αρχείο model.js

Δηλώσεις του παιχνιδιού πόκερ:

1. Δήλωση της Poker.Phases κλάσης που μοντελοποιεί το αντικείμενο phases ενός παιχνιδιού πόκερ.
2. Δήλωση της κλάσης Poker.Game (constructor function, prototype attributes and methods, static attributes and methods).

6.2.1 Poker.Phases

```
Poker.Phases = function () {
    _.bind(Game.Phases, this)(Poker.Phases.InitializationContext.phases);
};
_.extend(Poker.Phases.prototype, Game.Phases.prototype);
```

InitializationContex: Η phases περιέχει πέντε arrays ανάλογα με τις φάσεις στις οποίες βρίσκεται ο παίκτης. Στην φάση «start» τρέχει η onGameStart, όταν τελειώνει η φάση αυτή τρέχει Game.NF που βρίσκεται στο model.js του game και επιτρέπει στον παίκτη που βρίσκεται σε αυτή την φάση να πάει επομένη. Το ίδιο συμβαίνει και στις επόμενες τρείς φάσης. Ενώ στην τελευταία "End" φάση του παιχνιδιού τρέχει τις functions που υπάρχουν και στην Poker.Phases.onStart και αυτές που βρίσκονται στην Poker.Phases.onGameEnd.

```
Poker.Phases.InitializationContext = {
    phases: [
        { name: "Start",        onStart: Poker.Phases.onGameStart, onEnd: Game.NF, onPlayerAdvance: Poker.Phases.onPlayerAdvance },
        { name: "BetFirstTime", onStart: Poker.Phases.onStart,  onEnd: Game.NF, onPlayerAdvance: Poker.Phases.onPlayerAdvance},
        { name: "ChangeCards",  onStart: Poker.Phases.onStart,  onEnd: Game.NF, onPlayerAdvance: Poker.Phases.onPlayerAdvance },
        { name: "BetSecondTime", onStart: Poker.Phases.onStart,  onEnd: Game.NF, onPlayerAdvance: Poker.Phases.onPlayerAdvance },
        { name: "End",          onStart: _.compose(Poker.Phases.onStart, Poker.Phases.onGameEnd),  onEnd: Game.NF, onPlayerAdvance: Game.NF }
    ]
};
```

Η Poker.Phases είδαμε στην δήλωση της από πάνω ότι περιέχει τις εξής μεθόδους:

- onStart(): Δηλώνει στο session της CurrentPlayer στην αρχή του παιχνιδιού «NoPlayer» ή το name της CurrentPlayer. Επίσης, δηλώνει στο session της currentPhase το όνομα της φάσης που έχει η this.

```
onStart: function() {
    var currentPlayer = this.getCurrentPlayer();
    Session.set("currentPlayer", currentPlayer? currentPlayer.name: "NoPlayer");
    Session.set("currentPhase", this.getCurrentPhase().name);
},
```

- onGameStart(): Στο session του winner στην αρχή του παιχνιδιού υπάρχει null.

```
onGameStart: function() {
    Session.set("winner", null);
},
```

- onGameEnd() : Μέθοδος που ορίζει τον νικητή και τον χαμένο από την orderCardHolders που θα αναλύσουμε αργότερα. Ειδικότερα ο πρώτος παίκτης που βρίσκεται στην winnerOrder μπαίνει στην μεταβλητή winner της self και ο αλλός στην μεταβλητή losers της self. Παράλληλα προσθέτει και αφαιρεί το ποσό που παίχτηκε στην amount του wallet. Αρχικοποιεί την amountWon και για καθένα από τους χαμένους δημιουργεί μια function που προσθέτει το currentBet της loser στο amountWon . Στην συνέχεια αν ο χαμένος δεν είναι ο Host (υπολογιστής) στην μεταβλητή wallet μπαίνει η τιμή του wallet του χαμένου από την Wallets και αφαιρεί από εκεί το currentBet. Στην κονσόλα εμφανίζεται ο winner και στο session του winner αποθηκεύεται το όνομα του νικητή της μεταβλητής self. Αν πάλι δεν είναι νικητής ο Host προστίθεται στο wallet του winner το amountWon.

```
onGameEnd: function() {
    if (this.getCurrentPlayer() == null) {
        return;
    }
    var self = this;
    Session.set("currentPhase", this.getCurrentPhase().name);

    self.winnerOrder = Poker.Game.get(self.getCurrentPlayer().gameId).getWinnerOrderedPlayers();
    self.winner = _.first(self.winnerOrder);
    self.losers = _.rest(self.winnerOrder);
    var amountWon = 0;
    _.each(self.losers, function (loser) {
        amountWon += loser.currentBet;
        if (loser.name != "Host") {
            var wallet = Wallets.findOne({username: loser.name});
            Wallets.update({_id: wallet._id}, {$set : {amount: wallet.amount - loser.currentBet}});
        }
    });

    console.log(self.winner);

    Session.set("winner", self.winner.name);

    if (self.winner.name != "Host") {
        var wallet = Wallets.findOne({username: self.winner.name});
        Wallets.update({_id: wallet._id}, {$set : {amount: wallet.amount + amountWon}});
    }
}
);
```

- onPlayerAdvance() : Δηλώνει στο session της CurrentPlayer «NoPlayer» αν δεν υπάρχει παίκτης ή το name της CurrentPlayer. Επίσης, δηλώνει στο session της currentPhase το όνομα της φάσης που έχει η this.

```
onPlayerAdvance: function() {
    var currentPlayer = this.getCurrentPlayer();
    Session.set("currentPlayer", currentPlayer? currentPlayer.name: "NoPlayer");
},
```

6.2.2 Poker.Player

Μετά τον ορισμό των φάσεων του παιχνιδιού ας ορίσουμε και τους παίκτες του παιχνιδιού. Γνωρίζουμε ήδη ότι είναι δύο οι παίκτες σε αυτήν την εφαρμογή ο άνθρωπος και η μηχανή. Η κλάση Poker.Player μοντελοποιεί τους παίκτης του Poker.Game και κληρονομεί από την Card.Holder κλάση. Οι παράμετροι που παίρνει ο Poker.Player είναι οι εξής:

- id : ένα string id για το object του poker player.
- gameId : ένα game id του Poker.Game που παίζει αυτός ο παίκτης .
- deck : η τράπουλα του poker game αυτού του παίκτη. Κοινή παράμετρος για όλους τους παίκτες αυτού του παιχνιδιού.
- bin : ο κάτοχος των καρτών που απορρίφθηκαν . Κοινή παράμετρος για όλους τους παίκτες αυτού του παιχνιδιού.
- hiddenCards : είναι true αν αληθεύει η isTurned σημαία των καρτών αυτού του παίκτη / Card.Holder αυτό σημαίνει ότι ο UI θα δείξει το πίσω μέρος των καρτών του παίκτη.
- isHuman :Είναι true αν αληθεύει η isHuman σημαία του παίκτη. Αυτό σημαίνει ότι ο UI θα έχει τα ανθρώπινα στοιχεία αλληλεπίδρασης π.χ. κουμπιά, sing in κτλ.
- Ο Constructor της κλάσης Poker.Player κληρονομεί την κλάση Card.Holder.

```
Poker.Player = function(id, gameId, deck, bin, hiddenCards, isHuman) {
    _.bind(Card.Holder, this)(id, deck, bin);
    this.isHuman = isHuman;
    this.hiddenCards = hiddenCards;
    this.gameId = gameId;
    this.currentBet = 0;

    var specialized = isHuman? "Human" : "Engine";
    this.doChangeCards = Poker.Player[specialized].doChangeCards;
    this.doBet = Poker.Player[specialized].doBet;
};
```

Στο prototype του Poker.Player δημιουργούνται οι εξής μέθοδοι:

- ReplaceConditions(): Σε αυτή την μέθοδο αν η isCurrentPhase είναι ChangeCards, δηλαδή βρίσκεται στην φάση του παιχνιδιού που αλλάζεις φύλλα δημιουργούμε την μεταβλητή num2rep όπου η isPicked του getCards είναι true δηλαδή ο παίκτης έχει επιλέξει τα φύλλα που θέλει να αλλάξει τότε ή επιστρέφεται η μέθοδος return {}. Η άλλη μεταβλητή np που δημιουργείται καλείται σε περίπτωση που η isPicked του getCards είναι true. Τότε αν είναι 5 τα φύλλα που επιλέγει ο παίκτης να αλλάξει δεν επιστρέφει τίποτα γιατί απαγορεύεται από τους κανονισμού του παιχνιδιού. Επιπλέον επιτρέπεται να επιστρέψει 4 αν έχει έναν άσσο.

```
replaceConditions: function () {
    var self = this;

    if (self.isCurrentPlayer() &&
        Poker.Game.get(self.gameId).isCurrentPhase("ChangeCards")) {
        var num2rep = self.getCards({ isPicked: true }).length;
        switch (num2rep) {
            case 5:   return {};
            case 4:   var np = self.getCards({ isPicked: false });
                if (! np.length == 1 ||  np[0].rank != "A") {
                    return {};
                }
            default : return { isPicked: true };
        }

    } else {
        return {};
    }
},
```

- isCurrentPlayer(): Επιστρέφει true ή false ανάλογα με το αν ταιριάζει το name της this με αυτό του currentPlayer.

```
  isCurrentPlayer: function() {
      return Session.equals("currentPlayer", this.name);
  }
;
```

Τελειώνοντας και με τον σχολιασμό των prototypes του Poker.Player ας δούμε πως δημιουργούνται οι κλάσεις για τους δύο παίκτες.

1.)Human Player:

- doBet() : Αν η φάση του παιχνιδιού είναι BetFirstTime ή BetSecondTime τότε ο παίκτης ποντάρει και το currentBet του προστίθεται στην amount του. Στην μεταβλητή nextPlayer τοποθετείτε ο currentPlayer από την getCurrentPlayer. Στην συνέχεια ο παίκτης που δεν είναι Human δηλαδή η μηχανή, δέχεται το bet.

```
Poker.Player.Human = {
    /**
     * @param amount the amount a human player bet. Usually this method is invoked through a UI event.
     */
    doBet: function(amount) {
        var self = this;
        if (_.isUndefined(amount) || ! _.isNumber(amount)) {
            return;
        }
        self.currentBet += amount;

        Poker.Game.get(self.gameId).advanceCurrentPlayer();

        if (Poker.Game.get(self.gameId).isCurrentPhase("BetFirstTime")
            || Poker.Game.get(self.gameId).isCurrentPhase("BetSecondTime")) {
            var nextPlayer = Poker.Game.get(self.gameId).getCurrentPlayer();
            if (! nextPlayer.isHuman) {
                nextPlayer.doBet(amount);
            }
        }
    },
```

- doChangeCards(): Σε αυτή την μέθοδο χωρίς ορίσματα δημιουργούμε την replaceConditions που καλεί την replaceConditions() της self. Για καθεμιά από τις getCards(replaceCondition) της self δημιουργείται η function replace της self με ορίσματα cards, self.getDeck(), self.getBin(). Επίσης καλείται η advanceCurrentPlayer() του Poker.Game. Εδώ ο παίκτης Human μπορεί να εκτελέσει την doChangeCards αν βρίσκεται στην φάση του παιχνιδιού ChangeCards. Η μηχανή, στην συνεχεία , εκτελεί και αυτή την doChangeCards.

```
    doChangeCards: function() {
        var self = this;
        var replaceConditions = self.replaceConditions();

        if (replaceConditions.isPicked) {
            _.each(self.getCards(replaceConditions), function (card) {
                self.replace(card, self.getDeck(), self.getBin());
            });

            Poker.Game.get(self.gameId).advanceCurrentPlayer();

            if (Poker.Game.get(self.gameId).isCurrentPhase("ChangeCards")) {
                var nextPlayer = Poker.Game.get(self.gameId).getCurrentPlayer();
                if (! nextPlayer.isHuman) {
                    nextPlayer.doChangeCards();
                }
            }
        }
    }
```

2.)Engine Player:

- doBet(amount) : Η doBet της μηχανής είναι απλά να δεχτεί το currentBet του πραγματικού παίκτη. Εδώ amount είναι της μηχανής. Αυτή η μέθοδος εκτελείτε μετά το bet του παίκτη isHuman.

```
Poker.Player.Engine = {

    doBet: function(amount) {
        var self = this;
        if (_.isUndefined(amount)  || ! _.isNumber(amount)) {
            return;
        }
        self.currentBet += amount;

        Poker.Game.get(self.gameId).advanceCurrentPlayer();
    },
```

- doChangeCards(): Εδώ η μέθοδος αυτή εκτελεί την αλλαγή των φύλλων της μηχανής . Δημιουργείται το object cardsToChange που είναι αντικείμενο της discard() του PokerHandRating που θα αναλύσουμε αργότερα. Ορίζεται , επίσης, η delay στα 800 και για καθεμιά από τις κάρτες του cardsToChange επιπλέον 200ms. Τέλος με την _.bind δεσμεύεται η function _doChangeCards στο self object και αυτά καθυστερούν 2sec.

```
    doChangeCards: function() {
        var self = this;

        var cards = self.getCards({});
        var cardsToChange = new Poker.Rating(cards).discard();
        var delay = 800;

        _.each(cardsToChange, function (card) {
            _.delay(function() {
                Card.Holder.get(card.cardholderName).pick(card); }, delay+=200);
        });

        _.delay(_.bind(Poker.Player.Human.doChangeCards, self), 2000);
    }
```

6.2.3 Poker.Game

Μετά από τον ορισμό των παικτών του παιχνιδιού ορίζουμε το Poker.Game ως εξής:

```
Poker.Game = function (id) {
    _.bind(Poker.Phases, this)();

    var self = this;

    self.id = id;
    self.players = {};
    self.deck =  new Card.Holder("Deck_"+id, null, null, null, true);
    self.bin = new Card.Holder("Bin_"+id);

    self.deck.shuffle();

    Poker.Game._all[id] = self;
    Poker.Game._last = self;

    Poker.Game.Dep.changed();
};
_.extend(Poker.Game.prototype, Poker.Phases.prototype);
```

Μέσα στην μεταβλητή self δίνουμε όρισμα για την id, players, deck, bin και phases. Η self.deck και η self.bin είναι νέες κλάσεις της Card.Holder. Με την self.deck.shuffle() ανακατεύεται η τράπουλα για κάθε νέο παιχνίδι. Καλείται η changed() της Dep. Τέλος, στην prototype του Poker.Game προσθέτουμε την prototype του Poker.Phases.

Στην επόμενη εικόνα φαίνονται τα static fields και οι δηλώσεις των μεθόδων για την Poker.Game:

```
_.extend(Poker.Game, {
    _all: {},
    _last: null,
    Dep: new Deps.Dependency,
    get: function (id) {
        return Poker.Game._all[id];
    },
    getLast: function() {
        return Poker.Game._last;
    },
    crateEmpty: function() {
        var game = new Poker.Game('EmptyGame');
        game.startGame();
        return game;
    },
    createOnePlayerVsComputer: function(id) {
        if (Meteor.user() == null) {
            return;
        }

        var game = new Poker.Game(id);
        var host = game.createPlayer('Host', false, false);
        host.hide();
        var visitor = game.createPlayer(Meteor.user().username, false, true);

        game.acceptNewPlayer(visitor);
        game.acceptNewPlayer(host);

        game.startGame();

        return game;
    }
```

-Prototype πεδία και δηλώσεις μεθόδων της Poker.Game κλάσης:

Οι privates μέθοδοι που προσθέτουμε στο prototype της Poker.Game είναι:

- getPlayer(id): Κρατάει την id της μεταβλητής this του players.
- getPlayers():Κρατάει τις τιμές της this του players.
- createPlayer(id, hiddenCards, isHuman): Τα ορίσματα αυτή της μεθόδου είναι id, hiddenCards και isHuman. Ο newPlayer είναι αντικείμενο της Poker.Player και σε κάθε παίκτη δίνονται από την τράπουλα (deck) 5 φύλλα.

```
_.extend(Poker.Game.prototype, {
    getPlayer: function (id) {
        return this.players[id];
    },
    getPlayers: function () {
        return _.values(this.players);
    },
    createPlayer: function (id, hiddenCards, isHuman) {
        var self = this;
        var newPlayer = new Poker.Player(id, self.id, self.deck, self.bin, hiddenCards, isHuman);

        self.deck.pass(newPlayer, 5);

        if (hiddenCards) {
            newPlayer.hide();
        }

        self.players[id] = newPlayer;
        return newPlayer;
    },
```

- getWinnerOrderedPlayers(cardholders): Η rating είναι νέο αντικείμενο της Poker.Rating που θα αναλυθεί αργότερα. Εμφανίζεται στην κονσόλα το όνομα του cardholder, το order του(μεταβλητή που ορίζει πόσο δυνατός είναι ένας συνδυασμός και θα αναλύσουμε σε λίγο τον ορισμό της) και το όνομα rate που είναι το όνομα του συνδυασμού χαρτιών που έχει ο παίκτη

```
    getWinnerOrderedPlayers: function(cardholders) {
        var self = this;
        return _.sortBy(self.players, function (cardholder) {
            cardholder.rating = new Poker.Rating(cardholder.getCards());

            console.log("Poker.Game.getWinnerOrderedPlayers--------> "
                +cardholder.name+" "
                +cardholder.rating.order + "  "
                + cardholder.rating.rate.name);

            return cardholder.rating.order;
        });
    }
});
```

- Poker.Rating(cards): Η Poker.Rating ταξινομεί με την βοήθεια της _.sortBy τα φύλλα του κάθε παίκτη ανάλογα με το rankWeight και κατηγοριοποιεί τις κάρτες αναλόγως το rank και το suit. Επίσης βρίσκει τι συνδυασμό έχει ο παίκτης δηλαδή ένα έχει ζευγαρι, φουλ κτλ. Από την PokerHandRating.Rate και το επιστρέφει στην rate.is. Στο τέλος ορίζει την self.order που στην ουσία ανάλογα με τους κανόνες του παιχνιδιού ανακαλύπτει το πιο δυνατό φύλλο και κατ ‘ επέκταση τον νικητή.

```
Poker.Rating = function(cards) {
    var self = this;
    self.cards = _.sortBy(cards, function(card) { return card.rankWeight; });
    self.ranks = _.groupBy(this.cards, 'rank');
    self.suits = _.groupBy(this.cards, 'suit');

    self.rate = _.find(_.values(Poker.Rating.Rate), function(rate) {
        return rate.is(self);
    });
    self.order = 100000000000 * self.getOrder() + self.orderBetweenEquals();
};
```

Στην επόμενη εικόνα φαίνονται και οι υπόλοιπες δηλώσεις στο prototype της Poker.Game. Δηλωση της get και της is του CurrentPlayer και της CurrentPhase. Γενικότερα, μέσα ως prototype members ορίζονται οι παρακάτω μέθοδοι που βρίσκονται στην phases της this.

```
  getPlayer: function (id) {
      return this.players[id];
  },
  getPlayers: function () {
      return _.values(this.players);
  },
  getCurrentPlayer: function () {
      return this.phases.getCurrentPlayer();
  },
  getCurrentPhase: function() {
      return this.phases.getCurrentPhase();
  },
  isCurrentPhase: function(phaseName) {
      return this.phases.isCurrentPhase(phaseName);
  },
  isCurrentPlayer: function(player) {
      return this.phases.isCurrentPlayer(player);
  },
  advanceCurrentPlayer: function(phaseName) {
      return this.phases.advanceCurrentPlayer(phaseName);
  },
  acceptNewPlayer: function(player) {
      return this.phases.acceptNewPlayer(player);
  },
  startGame: function() {
      return this.phases.startGame();
  }
;
```

Prototypes member of PokerHandRating:

1.)get: Περιέχει τις τιμές του rate.
2.)getOrder : Οι κανόνες του παιχνιδιού, δηλαδή ποιοι συνδυασμοί υπάρχουν και ποιος συνδιασμός είναι πιο δυνατός.
3.)OrderBetweenEquals: Κανόνες για το ποιός κερδίζει αν έχουν τον ίδιο συνδυασμό.
4.)getName: Το name της rate.
5.)hasAce: Αν υπάρχει άσσος στα 5 φύλλα.
6.)hasOfOneKind: Μέθοδος που αναγνωρίζει πόσες φορές υπάρχει το ίδιο rank.
7.)getOfOneKind: Μέθοδος που επιστρέφει τα φύλλα με το ίδιο rank.
8.)hasOfOneSuit: Επιστρέφει πόσα φύλλα υπάρχουν με το ίδιο suit.
9.)getOfOneSuit: Επιστρέφει πόσα ζευγάρια υπάρχουν.
10.)hasInARow: Ανάλογα το rankWeight βρίσκει αν τα φύλλα είναι στην σειρά (κέντα). Αυτό το κάνει τοποθετόντας στην μεταβλητή packOfN το πρώτο φύλλο της κάρτας και εξατάζοντας για όλα τα φύλλα την φθίνουσα ακολουθία του rankWeight. Αν είναι στην σειρά επιστρέφει true διαφορετικά false.
11.)discart: Μέθοδος για να αλλάξει τα φύλλα ο υπολογιστής.

```
_.extend(Poker.Rating.prototype, {
    get: function () {
        return this.rate;
    },
    getOrder: function () {
        return this.rate.order;
    },
    orderBetweenEquals: function () {
        return this.rate.orderBetweenEquals(this);
    },
    getName: function () {
        return this.rate.name;
    },
    hasAce: function () {
        return _.find(this.cards, function(card) { return card.rank == "A"; }) != null;
    },
    hasOfOneKind: function (n) {
        var times = 0;
        _.each(_.values(this.ranks), function (rank) { if (rank.length == n) times++; });
        return times;
    },
    getOfOneKind: function (n) {
        var times = [];
        _.each(_.values(this.ranks), function (rank) { if (rank.length == n) times.push(rank[0]); });
        return times;
    },
    hasOfOneSuit: function (n) {
        return _.some(_.values(this.suits), function (suit) { return suit.length == n; });;
    },
    getOfOneSuit: function (n) {
        return _.filter(_.values(this.suits), function (suit) { return suit.length == n; });
    },
```

```
    },
    hasInARow: function (n) {
        var cards = this.cards;
        while (cards.length >= n) {
            var packOfN = _.first(cards, n);
            var prev = packOfN[0].rankWeight - 1;
            if (_.every(packOfN, function(c) { return (++prev) == c.rankWeight; })) {
                return true;
            }
            cards = _.rest(cards);
        }
        return false;
    },
    discard: function () {
        return this.rate.discard(this);
    }
});
```

Η Poker.Rating.Rate είναι η μέθοδος μέσω της οποίας καταλαβαίνει ο υπολογιστής τι συνδυασμό φύλλων έχει ο εκάστοτε παίκτης. Η μεταβλητή order παίρνει ορίσματα ανάλογα με τον συνδιασμό. Η δήλωση ξεκινάει με το δυνατότερο φύλλο όπου το order έχει την τιμή 1 και συνεχίζει η αρίθμηση του order μα αύξουσα σειρά . Συνεπώς όσο μικρότερη τιμή έχει η order τόσο πιο δυνατός είναι ο συνδυασμός φύλλων. Ας δούμε πως δηλώνονται οι συνδυασμοί φύλλων:

- RoyalFlush: Θυμίζουμε ότι ο συνδυασμός αυτός είναι 5 φύλλα στην σειρά ίδιου suit με άσσο. Αν ο υπολογιστής διαθέτει αυτό το φύλλο φυσικά δεν κάνει καμιά αλλαγή. Και αν και οι δύο που αναμετρούνται έχουν το ίδιο φύλλο δεν επιστρέφεται τίποτα – ισοπαλία. Αν ο υπολογιστής έχει αυτό το φύλλο προφανώς δεν αλλάζει τίποτα.

```
RoyalFlush:  {
    name: "RoyalFlush",
    order: 1,
    is: function(hand) {
        return hand.hasInARow(5)
            && hand.hasOfOneSuit(5)
            && hand.hasAce();
    },
    discard: function(hand) {
        return [];
    },
    orderBetweenEquals: function (hand) {
        return 0;
    }
},
```

- StraightFlush: Η διαφορά αυτού του συνδυασμού με τον από πάνω είναι ότι εδώ δεν είναι απαραίτητο να υπάρχει άσσος , αρκεί να είναι πέντε φύλλα στην σειρά ίδιου χρώματος και είναι ο δεύτερος καλύτερος συνδυασμός στο poker (order=2) . Στην orderBetweenEquals επιστρέφεται το βάρος του μεγαλύτερου φύλου. Αν δηλαδή και οι δύο παίκτες έχουν order:2 αυτός που το βάρος του πρώτου φύλου της κέντας είναι μεγαλύτερο αυτός είναι ο νικητής. Ούτε εδώ ο υπολογιστής αλλάζει φύλλα.

```
StraightFlush: {
    name: "StraightFlush",
    order: 2,
    is: function(hand) {
        return hand.hasInARow(5)
            && hand.hasOfOneSuit(5);
    },
    discard: function(hand) {
        return [];
    },
    orderBetweenEquals: function (hand) {
        return _.first(hand.cards).rankWeight;
    }
},
```

- FourOfKind : Το γνωστό καρέ. Όταν στα πέντε φύλλα τα τέσσερα είναι ίδια. Ούτε σε αυτήν την περίπτωση ο υπολογιστής κάνει αλλαγές. Αν και οι δύο παίκτες έχουν καρέ κερδίζει αυτός που το έχει στο μεγαλύτερο φύλο , δηλαδή το βάρος (weight) του φύλου του είναι μεγαλύτερο .Επειδή είναι το τρίτο καλύτερο φύλλο του πόκερ και να κάνει αλλαγή ο υπολογιστής το πέμπτο φύλλο δεν γίνεται να προκύψει καλύτερος συνδυασμός. Οπότε ούτε εδώ αλλάζει φύλλα ο υπολογιστής.

```
FourOfAKind: {
    name: "FourOfAKind",
    order: 3,
    is: function(hand) {
        return hand.hasOfOneKind(4);
    },
    discard: function(hand) {
        return [];
    },
    orderBetweenEquals: function (hand) {
        return hand.getOfOneKind(1)[0].rankWeight;
    }
},
```

- FullHouse: τρία όμοια φύλλα και δύο όμοια. Πάλι ο υπολογιστής δεν θα κάνει αλλαγή. Αν και οι δύο συμμετέχοντες του παιχνιδιού έχουν φουλ θα νικήσει την παρτίδα αυτός που το βάρος της τριάδας θα είναι μεγαλύτερο.

```
FullHouse: {
    name: "FullHouse",
    order: 4,
    is: function(hand) {
        return hand.hasOfOneKind(3) && hand.hasOfOneKind(2);
    },
    discard: function(hand) {
        return [];
    },
    orderBetweenEquals: function (hand) {
        return hand.getOfOneKind(3)[0].rankWeight;
    }
},
```

- Flush : Το γνωστό χρώμα όταν και τα πέντε φύλλα είναι στο ίδιο χρώμα . πχ Πέντε φύλλα που να είναι καρό. Ούτε εδώ ο υπολογιστής δεν θα αλλάξει φύλλα αφού πρόκειται για ένα πολύ καλό συνδυασμό και έχει πολλές πιθανότητες να κερδίσει . Σε περίπτωση που και οι δύο παίκτες έχουν χρώμα κερδίζει αυτός με το μεγαλύτερο φύλλο στο χρώμα.

```
Flush: {
    name: "Flush",
    order: 5,
    is: function(hand) {
        return hand.hasOfOneSuit(5);
    },
    discard: function(hand) {
        return [];
    },
    orderBetweenEquals: function (hand) {
        return hand.getOfOneSuit(5)[0].rankWeight;
    }
},
```

- Straight : Η κέντα , όταν πέντε φύλλα είναι στην σειρά . Ο υπολογιστής δεν θα αλλάξει φύλλο γιατί και αυτός είναι καλός συνδυασμός. Κερδίζει ο παίκτης έχει κέντα στο μεγαλύτερο φύλλο. Ο υπολογιστής δεν κάνει καμία αλλαγή γιατί είναι πολύ καλός συνδυασμός και δεν θα του έρθει κάτι καλύτερο.

```
Straight: {
    name: "Straight",
    order: 6,
    is: function(hand) {
        return hand.hasInARow(5);
    },
    discard: function(hand) {
        return [];
    },
    orderBetweenEquals: function (hand) {
        return hand.cards[0].rankWeight;
    }
```

- ThreeOfKind: Στα πέντε φύλλα τα τρία να είναι όμοια. Ο υπολογιστής θα αλλάξει το ένα από τα υπόλοιπα δύο (getofOneKind(1) τις μονοφυλλίες). Κερδίζει ο παίκτης με το μεγαλύτερο rankWeight στην τριάδα του.

```
,,
ThreeOfAKind: {
    name: "ThreeOfAKind",
    order: 7,
    is: function(hand) {
        return hand.hasOfOneKind(3) && hand.hasOfOneKind(1) == 2;
    },
    discard: function(hand) {
        return hand.getOfOneKind(1);
    },
    orderBetweenEquals: function (hand) {
        return hand.getOfOneKind(3)[0].rankWeight;
    }
```

- TwoPair : Από τα πέντε φύλλα τα δύο είναι ζευγάρια . Ο υπολογιστής θα αλλάξει την μονοφυλλία του διότι έχει αρκετές πιθανότητες να σχηματίσει φουλ. Κερδίζει ο παίκτης που το ζευγάρια του είναι πιο δυνατά ανάλογα το rankWeight. Αν έχουν κοινό το ένα από τα δύο ζευγάρια , τότε κερδίζει αυτός που το δεύτερο ζευγάρι του είναι πιο δυνατό. Ο υπολογιστής καταλαβαίνει πιο φύλλο είναι δυνατότερο ως εξής πολλαπλασιάζει το πρώτο φύλλο ζευγάρι με το μεγαλύτερο rankWeight με το 100 και σε αυτό προσθέτει το δεύτερο . Στην μέθοδο Poker.Rating υπάρχει η εντολή self.order = 100000000000 * self.getOrder() + self.orderBetweenEquals();
  Οπότε η self.order ορίζεται ως έναν αριθμό οπου του πολλαπλασιάζεις την getOrder() και σε αυτόν προσθέτεις την self.orderBetweenEquals(). Ας δώσουμε ένα παράδειγμα. Έστω ότι ο πρώτος παίκτης έχει δύο ζευγάρια του 6 και του 10 και το δεύτερος παίκτης έχει ζευγάρια του 8 και του 10. Σε αυτήν τη παρτίδα και οι δύο παίκτες έχουν order:8.Το rankWeight του 10 είναι 5, του 8 είναι 7 και του 6 είναι 9. Αρα, η self.order του πρώτου θα είναι 100000000000*8+100*10+9=800000001009 ενώ η self.order του δεύτερου θα είναι 100000000000*8+100*10+7= 800000001007. Νικητής είναι αυτός με την μικρότερη self.order.

```
TwoPair: {
    name: "TwoPair",
    order: 8,
    is: function(hand) {
        return hand.hasOfOneKind(2) == 2;
    },
    discard: function(hand) {
        return hand.getOfOneKind(1);
    },
    orderBetweenEquals: function (hand) {
        var first = hand.getOfOneKind(2)[0].rankWeight;
        var second = hand.getOfOneKind(2)[1].rankWeight;
        return 100*first + second;
    }
}
```

- OnePair : Στα πέντε φύλλα το ένα να είναι ζευγάρι . Εδώ αν και οι δύο διαθέτουν από ένα ζευγάρι κερδίζει αυτός με το πιο δυνατό ζευγάρι (μεγαλύτερο rankWeight) ,σε περίπτωση που έχουν το ίδιο ζευγάρι ο νικητής κρίνεται ανάλογα με το επόμενο μεγαλύτερο φύλλο που έχει. Η εφαρμογή το αναγνωρίζει αυτό με τον τρόπο που είδαμε και πάνω δηλαδή πολλαπλασιάζοντας με το 100 το ζευγάρι και προσθέτοντας σε αυτό τις μονοφυλλίες. Αν ο υπολογιστής έχει ένα ζευγάρι θα αλλάξει τις τρείς μονοφυλλίες του περιμένοντας να του έρθει η τριάδα η καρέ, ώστε το φύλλο του να γίνει πιο δυνατό (μεγαλύτερο order).

```
OnePair: {
    name: "OnePair",
    order: 9,
    is: function(hand) {
        return hand.hasOfOneKind(2);
    },
    discard: function(hand) {
        return hand.getOfOneKind(1);
    },
    orderBetweenEquals: function (hand) {
        var first = hand.getOfOneKind(2)[0].rankWeight;
        var second = hand.getOfOneKind(1)[0].rankWeight;
        return 100*first + second;
    }
},
```

- HighCard: Στα πέντε φύλλα δεν υπάρχει κανείς από τους προηγούμενους συνδυασμούς. Όταν στα πέντε φύλλα δεν υπάρχει κανένα όμοιο λαμβάνουμε υπόψη το βάρος του φύλλου. Κρατάει ο υπολογιστής το μεγαλύτερο και αλλάζει τα υπόλοιπα. Επιστρέφει τα βάρη των φύλλων από το πιο δυνατό έως το πιο αδύνατο , δίνοντας μεγαλύτερη βαρύτητα στο μεγαλύτερο αυτό δηλαδή με το μεγαλύτερο rankWeight. Εδώ όπως φαίνεται και στον κώδικα το rankWeight του μεγαλύτερου φύλλου το πολλαπλασιάζει με το 10000 το αμέσως μικρότερο με το 1000 , το ακόμα μικρότερο με το 100 κτλ. Στο τέλος προκύπτει ένα άθροισμα . Το άθροισμα αυτό προστίθεται στο 10(order)*100000000000 και έτσι καταλήγουμε στον αριθμό της self.order σύμφωνα με την οποία κερδίζει η μικρότερη τιμή.

```
},
HighCard: {
    name: "HighCard",
    order: 10,
    is: function(hand) {
        return hand.hasOfOneKind(1) == 5;
    },
    discard: function(hand) {
        return _.first(hand.getOfOneKind(1), 3);
    },
    orderBetweenEquals: function (hand) {
        var oneOfAKind = hand.getOfOneKind(1);
        console.log("orderBetweenEquals -----> "+ _.values(oneOfAKind[0]));
        return 10000*oneOfAKind[0].rankWeight
                + 1000*oneOfAKind[1].rankWeight
                + 100*oneOfAKind[2].rankWeight
                + 10*oneOfAKind[3].rankWeight
                + oneOfAKind[4].rankWeight;
    }
}
```

6.3 Αρχείο poker.js

Το poker.js στην ουσία δένει το poker.html με το model.js.. Το poker.html είναι αυτό που βλέπει ο χρήστης , το αρχείο model.js είναι αυτό που αντιλαμβάνεται η μηχανή , το λειτουργικό κομμάτι του poker. Και τέλος, το αρχείο poker.js ενώνει το εικονικό με το λειτουργικό κομμάτι του poker. Εδώ δημιουργούνται οι εξής πέντε μέθοδοι :

- PlayerName(): Επιστρέφει το όνομα του παίκτη της μεταβλητής this.

```
Template.pokerPlayer.playerName = function () {
    return this.name;
};
```

- canChangeCards(): Αν ο παίκτης είναι άνθρωπος (isHuman) και το session της currentPlayer είναι το όνομα που είναι στην μεταβλητή this και το session το currentPhase είναι το ChangeCards. Όταν συμβαίνουν αυτά λειτουργεί το κουμπί «Change Cards» πού είναι στα δεξιά των καρτών του παίκτη.

```
Template.pokerPlayer.canChangeCards = function () {
   return this.isHuman && Session.equals("currentPlayer", this.name) //this.isCurrentPlayer()
      && Session.equals("currentPhase", "ChangeCards") //Poker.Game.get(this.gameId).isCurrentPhase("ChangeCards")
};
```

- canBet(): Αν ο παίκτης είναι άνθρωπος (isHuman) και το session της currentPlayer είναι το όνομα που είναι στην μεταβλητή this και το session το currentPhase είναι το BetFirstTime ή το session το currentPhase είναι το BetSecondTime. Όταν συμβαίνουν αυτά λειτουργεί το κουμπί μέσω του οποίου μπορεί να κάνει bet ο παίκτης.

```
Template.pokerPlayer.canBet = function () {
    return this.isHuman == true
        && Session.equals("currentPlayer", this.name)
        && (Session.equals("currentPhase", "BetFirstTime")
            || Session.equals("currentPhase", "BetSecondTime"));
};
```

- isWinner(): επιστρέφει το όνομα της this που είναι στο session του winner. Όταν συμβαίνει αυτό πρασινίζει το όνομα του παίκτη στα αριστερά του τραπεζιού.

```
Template.pokerPlayer.isWinner = function () {
    return Session.equals("winner", this.name);
};
```

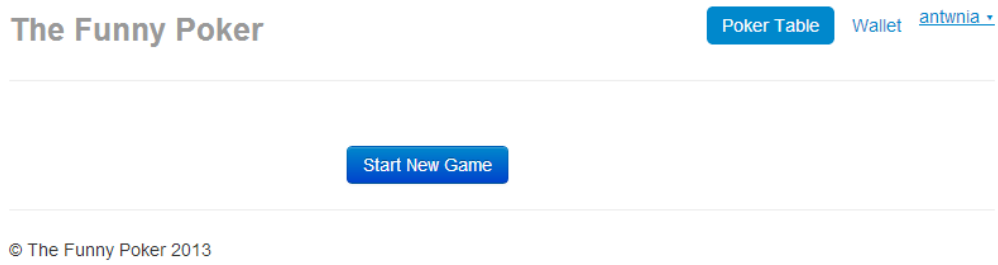
- events: Με το πάτημα του κουμπιού poker-change-cards καλείται η doChangeCards. Ενώ με click στο li.poker-bet-X' για amount X εκτελείται player.doBet(X).

```
Template.pokerPlayer.events({
    'click button.poker-change-cards': function () {
        this.doChangeCards();
    },
    'click li.poker-bet-0': function () {
        this.doBet(0);
    },
    'click li.poker-bet-10': function () {
        this.doBet(10);
    },
    'click li.poker-bet-20': function () {
        this.doBet(20);
    },
    'click li.poker-bet-50': function () {
        this.doBet(50);
    },
    'click li.poker-bet-200': function () {
        this.doBet(200);
    }
});
```

ΚΕΦΑΛΑΙΟ 6: ΑΝΑΛΥΣΗ ΤΗΣ MAIN

Στο αρχείο main βρίσκεται ο κώδικας της κεντρικής ιστοσελίδας. Αυτό το αρχείο τρέχει στον client και περιέχει την main.html η main.js και την main.css.

6.1 main.html



Εικόνα6.1 : Τι βλέπει ο χρήστης όταν τρέχει η main.html

Δημιουργία του main.html με την βοήθεια του bootstrap:

```
<body>
<div class="container-fluid">
    <div class="masthead">
        {{>navigation}}
    </div>

    <div class="row-fluid">
        {{>mainSection}}
    </div>

    <div class="row-fluid">
        <hr>
        <div class="footer">
            <p>&copy; The Funny Poker 2013</p>
        </div>
    </div>
</div>
</body>
```

Βλέπουμε ότι καλούνται οι Navigation και mainSection. Ενώ παράλληλα δημιουργείται η footer.

- navigation: Δημιουργείτε το nav επιλογες Poker Table και Wallet πάνω αριστερά αν ο χρήστης έχει κάνει login, αλλιώς εμφανίζεται η φόρμα του login. Επιπλέον φαίνεται ο τίτλος της ιστοσελίδας πάνω δεξιά "The funny Poker".

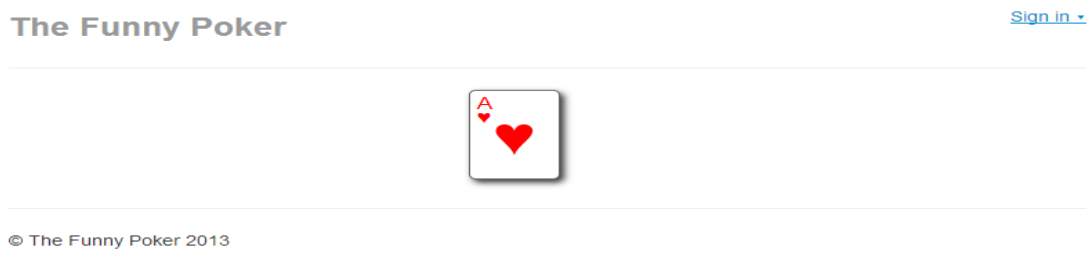
Εικόνα6.1 template navigation

```
<template name="navigation">
    <ul class="nav nav-pills pull-right">
        {{#if currentUser}}
            <li class="nav-poker {{pokerStatus}}"><a href="#">Poker Table</a></li>
            <li class="nav-wallet {{walletStatus}}"><a href="#">Wallet</a></li>
        {{/if}}
            <li>{{loginButtons align="right"}}</li>
    </ul>
    <h3 class="muted">The Funny Poker</h3>
    <hr>
</template>
```

- mainSection: Ο εκάστοτε παίκτης που έχει την ιδιότητα currentUser είναι isPokerActive και isWalletActive δηλαδή έχει πρόσβαση στο παιχνίδι (game) και στο wallet του αν δεν είναι currentUser έχει πρόσβαση στο welcomeView.

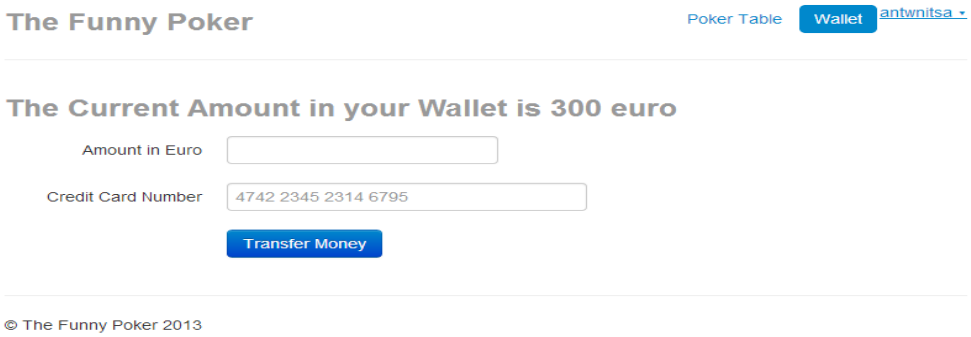
```
<template name="mainSection">
    {{#if currentUser}}
        {{#if isPokerActive}}
            {{>game}}
        {{/if}}
        {{#if isWalletActive}}
            {{>wallet}}
        {{/if}}
    {{else}}
        {{>welcomeView}}
    {{/if}}
</template>
```

- welcomeView: Στην αρχική σελίδα πριν ο χρήστης κάνει sign in φαίνεται ένας άσσος καρδιά.



Εικόνα6.2 Template welcomeView

- wallet:Στην αρχή εμφανίζεται το ποσό που διαθέτει ο χρήστης στο Wallet του(κάθε καινούργιος παίκτης ξεκινά με 300 ευρώ). Εδώ εικονίζονται ,επίσης, δύο πεδία. Πρώτα από όλα το Amount in Euro που δέχεται το πόσο σε ευρώ και το προσθέτει στο wallet και δεύτερο το πεδίο Credit Card Number όπου υπάρχει ο αριθμός μια εικονικής πιστωτικής κάρτας. Κάτω από αυτά τα πεδία δημιουργείται το κουμπί "Transfer Money" για την μεταφορά χρημάτων από την πιστωτική στο wallet.




```
<template name="wallet">
    <h3 class="muted">The Current Amount in your Wallet is {{amount}} euro</h3>

    <form class="form-horizontal">
        <div class="control-group">
            <label class="control-label" for="addedAmount">Amount in Euro</label>
            <div class="controls">
                <input type="text" id="addedAmount"/>
            </div>
        </div>
        <div class="control-group">
            <label class="control-label" for="credidCardNumber">Credit Card Number</label>
            <div class="controls">
                <input type="text" class="span6" id="credidCardNumber" placeholder="4742 2345 2314 6795"/>
            </div>
        </div>
        <div class="control-group">
            <div class="controls">
                <button type="submit" class="btn btn-primary">Transfer Money</button>
            </div>
        </div>
    </form>

</template>
```

- game: Κάθε παίκτης είναι pokerPlayer και εκτελεί gameActions.

```
<template name="game">
    {{#each players}}
    {{>pokerPlayer}}
    {{/each}}
    {{>gameActions}}
</template>
```

- gameActions: Όταν τελείωση το παιχνίδι δίνεται η επιλογή στο παίκτη να ξεκινήσει καινούργιο πατώντας το κουμπί “Start New Game”.


```
<template name="gameActions">
    {{#if gameOver}}
    <div class="offset4">
        <p> </p>
        <button class="btn btn-primary">Start New Game</button>
    </div>
    {{/if}}
</template>
```

6.2 main.css

Το CSS αρχείο της main:

```
body {
     margin: 2em 20%;
}
.options.active {
     position: fixed;
     top: 1em;
     right: 1em;
     background: #ddd;
     padding: .5em;
}
.options.active h3 {
     font-size: 1.2em;
}
.options.active ul {
     padding: 0;
}
.options.active li {
     color: #00c;
     text-decoration: underline;
     margin-left: 1.5em;
     cursor: pointer;
}
.options.active li:hover {
     text-decoration: none;
}
div.clear {
     clear: both;
     height: 0;
     line-height: 0;
     font-size: 1px;
     visibility: hidden;
}

.nav-poker li { }
.nav-wallet li { }
```

6.2 main.js

Ο js κώδικας της main. Εδώ πρώτα από όλα το user registration and application access Management της Meteor Accounts package. Χρησιμοποιήσαμε το πακέτο αυτό που απαιτεί username και προαιρετικά email.

```
Accounts.ui.config({
    passwordSignupFields: 'USERNAME_AND_OPTIONAL_EMAIL'
});
```

Δημιουργία ενός κενού παιχνιδιού που θα χρησιμοποιηθεί στην εφαρμογή αυτή για την εικόνα του χρήστη μέχρι να ενεργοποιηθεί το κουμπί «Start Game» .

```
Poker.Game.crateEmpty();
```

Στον παρακάτω κώδικα φαίνεται πως μοντελοποιούνται τα templates game και gameActions του main.html κώδικα .

```
Template.game.players = function() {
    Poker.Game.Dep.depend();
    return Poker.Game.getLast().getPlayers();
};

Template.gameActions.gameOver = function() {
    return Session.equals("currentPhase", "End");
};
```

Ο επόμενος κώδικας είναι του Wallet τα view templates και οι δηλώσεις των wallets events. Βλέπουμε ότι η amount συσχετίζεται με το username της βάσης δεδομένων user.

```
Template.wallet.amount = function() {
    return Wallets.findOne({username:Meteor.user().username}).amount;
};

Template.wallet.events({
    'click button': function () {
        var wallet = Wallets.findOne({username:Meteor.user().username})
        var newAmount = _.isNumber(wallet.amount) ? wallet.amount : parseInt(wallet.amount);
        newAmount += parseInt(document.getElementById("addedAmount").value); // $("#addedAmount").val()
        Wallets.update({_id: wallet._id}, {$set : {amount: newAmount}});
    }
});
```

Τέλος, φαίνονται τα navigation και mainSection templates. Το poker και το wallet λειτουργούν όταν το pokerStatus και το walletStatus είναι active. Με click στο nav-poker γίνεται active το pokerStatus και με click στο nav-wallet γίνεται active το walletStatus.

```
Template.mainSection.isPokerActive = function() {
    return Session.equals("pokerStatus", "active");
};

Template.mainSection.isWalletActive = function() {
    return Session.equals("walletStatus", "active");
};

Template.navigation.walletStatus = function() {
    return Session.get("walletStatus");
};

Template.navigation.pokerStatus = function() {
    return Session.get("pokerStatus");
};

Template.navigation.events({
    'click li.nav-poker': function () {
        Session.set("walletStatus", "");
        Session.set("pokerStatus", "active");
    },
    'click li.nav-wallet': function () {
        Session.set("walletStatus", "active");
        Session.set("pokerStatus", "");
    }
});
```

ΚΕΦΑΛΑΙΟ 7: ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΤΑΣΕΙΣ ΑΝΑΠΤΥΞΗΣ

Στο κεφάλαιο αυτό συνοψίζουμε σύντομα κάποια χρήσιμα συμπεράσματα της εφαρμογής που αναπτύχθηκε καθώς και κάποιες κατευθύνσεις για μελλοντικές επεκτάσεις της.

Συμπερασματικά βλέπουμε ότι πλέον έχουν δημιουργηθεί νέες τεχνολογίες η οποίες καθιστούν ευκολότερη την δημιουργία web εφαρμογών αφού μπορούν να διαχειριστούν και να συνδέσουν βάση δεδομένων με app server και browser. Επιπλέον, η δημιουργία αυτής της εφαρμογής γίνεται πιο διασκεδαστική χρησιμοποιώντας το bootstrap για τα css αρχεία και τη βιβλιοθήκη undercore.js που "λύνει" στην πραγματικότητα τα χέρια του προγραμματιστή , αφού προσφέρει πολλές εντολές με τις οποίες αποφεύγονται οι βρόγχοι loop της javascript.

Ένα από τα χρησιμότερα εργαλεία που χρησιμοποιήθηκαν για την επεκτασιμότητα αυτής της εφαρμογής είναι το MVC (Model-View-Controler) που διαχωρίζει τα δεδομένα από τη λογική και την παρουσίαση. Ουσιαστικά δημιουργείται μία εφαρμογή η οποία έχει τρία επίπεδα, το επίπεδο των models , το επίπεδο των controllers και το επίπεδο των views και το κάθε επίπεδο επιτελεί ξεχωριστό έργο και ταυτόχρονα συνεργάζεται με τα άλλα επίπεδα. Η funny poker είναι μια σωστή MVC εφαρμογή αφού τα τρία επίπεδα του poker και του cards είναι ξεκάθαρα καθορισμένα και δεν συμπλέκονται.

Ξεχωρίζοντας τις κάρτες από το υπόλοιπό παιχνίδι , η κάρτες αυτές μπορούν αργότερα να χρησιμοποιηθούν για την δημιουργία άλλων παιχνιδιών , όπως μπιρίμπα ή πόκα. Το ίδιο συμβαίνει και με το αρχείο πόκερ που περιέχει γενικότερα τους κανόνες του πόκερ και έτσι θα μπορούσε να χρησιμοποιηθεί για την δημιουργία μιας εφαρμογής texas holdem πόκερ. Τέλος το αρχείο game που περιέχει τις φάσεις της συγκεκριμένης παραλλαγής του πόκερ θα μπορούσε να χρησιμοποιηθεί σε άλλη πλατφόρμα παιχνιδιών.

Θα μπορούσαμε να βελτιώσουμε στην εφαρμογή τον τρόπο με τον οποίο η μηχανή επιλέγει τα φύλλα που θα κρατήσει. Προφανώς στους συνδυασμούς που αποτελούνται από πέντε φύλλα όπως είναι το φλος ρουαγιαλ, στρειτ φλος, χρώμα , φουλ και κέντα η μηχανή δεν θα κάνει καμιά αλλαγή. Όταν όμως έχει τριάδα, δύο ζεύγοι ή ζεύγος η μηχανή θα αλλάξει τις μονοφυλλίες. Αυτή η αλλαγή στηρίζεται στο γεγονός ότι αυτοί οι συνδυασμοί είναι αρκετά ισχυροί όταν πρόκειται για δύο αντιπάλους. Ωστόσο, θα μπορούσαμε να προσθέσουμε στον κώδικα εντολές για να κρατήσει η μηχανή σε αυτούς τους συνδυασμούς και το μεγαλύτερο φύλλο από τις μονοφυλλίες. Ακόμα καλύτερα, θα μπορούσαμε να ορίσουμε τον τρόπο που επιλέγει φύλλα ο υπολογιστής μέσω πράκτορα και τεχνητής νοημοσύνης. Στόχος την διπλωματικής είναι η ανάπτυξη λογισμικού για on-line πόκερ αλλά όχι η μελέτη ευφυών τρόπων (αλγορίθμων και ευριστικών) για πόκερ.

Ας δούμε ένα παράδειγμα, έστω ότι η μηχανή θα έχει 7, 8, 9, 10, K . Σε αυτήν την εφαρμογή ο υπολογιστής θα κρατούσε το 10 και τον K που είναι τα μεγαλύτερα φύλλα και θα άλλαζε τα υπόλοιπα τρία φύλλα. Αν χρησιμοποιούσαμε πράκτορα ίσως θα άλλαζε τον K , περιμένοντας 6 ή J για να σχηματιστεί κέντα. Έτσι , ο πράκτορας θα μπορούσε να λειτουργεί σύμφωνα με πιθανότητες και η μηχανή θα ήταν πιο ανταγωνιστική.

Προτάσεις για προέκταση της εφαρμογής:

1.) Μια πιθανή προέκταση είναι να μπορούν παίζουν αντίπαλοι δύο χρήστες. Με τον τρόπο που είναι χτισμένος ο κώδικας αυτό είναι εύκολα εφικτό με ελάχιστες τροποποιήσεις.
2.) Θα μπορούσαμε επίσης στην σελίδα αυτή να προσθέσουμε κι άλλα παιχνίδια με τράπουλα. Μετά την εισαγωγή του χρήστη να επιλέγει τι παιχνίδι επιθυμεί να παίξει.

Βιβλιογραφία:

http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

http://underscorejs.org/

http://meteor.com/

http://ui.canjs.us/

http://www.mongodb.org/

http://www.nodebeginner.org/

http://twitter.github.io/bootstrap/