ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
"ΔΙΔΑΚΤΙΚΗ ΣΤΗ ΤΕΧΝΟΛΟΓΙΑ &
ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑ – ΔΙΚΤΥΟΚΕΝΤΡΙΚΑ
ΣΥΣΤΗΜΑ"

Μελέτη, ανάλυση & προτάσεις χρήσης και αξιοποίησης των PaaS και SaaS του Google Cloud

Όνομα: Αλέξανδρος
Επώνυμο: Δουλγκέρης
ΑΜ: ΜΕ/09050

Επιβλέπων καθηγητής: Απόστολος Μηλιώνης

Οκτώβριος 2011

Περιεχόμενα

Πρόλογος

Η παρακάτω εργασία αποτελεί μια παρουσίαση του Google App Engine και του Google Web Toolkit. Σκοπός πέρα απο την θεωρητική κάληψη των παραπάνω τεχνολογιών είναι και η παροχή μικρών παραδειγμάτων που υποδεικνύουν την χρήση τους.

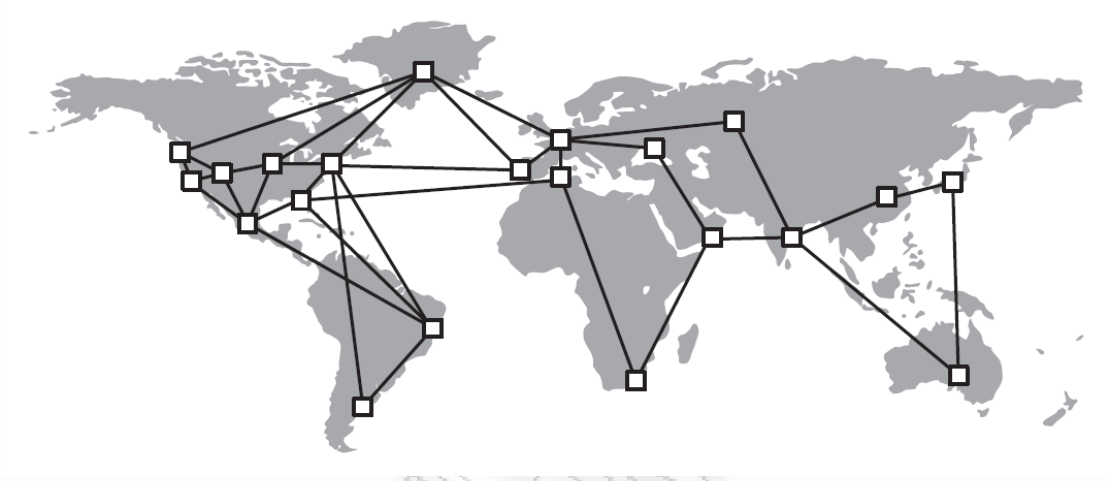
Επίσης υπάρχει και ξεχωριστή αναφορά στην συνεργασία (Integrarion) του Google Web Toolkit με το Seam Framework με σκοπό να δούμε πως μπορούν αυτές οι δύο τεχνολογίες να επικοινωνήσουν για να μπορέσουμε να φτιάξουμε εφαρμογές στις οποίες η βάση δεδομένων δεν θα είναι στο Google Infrastructure αλλά όπου θέλουμε εμείς. Αυτό μας δίνει επίσης την δυνατότητα να χρησιμοποιήσουμε οποιαδήποτε βάση δεδομένων επιθυμούμε (Oracle, MySQL, κλπ.).

Τέλος παρουσιάζεται και μια σημαντική δυνατότητα που έχει αναπτύξει η Google και ονομάζεται Secure Data Connector για να μπορούν οι εφαρμογές που είναι στην υποδομή (infrastructure) της να επικοινωνούν με εξωτερικές εφαρμογές μέσω διαδικτύου. Στα πλαίσια αυτού του κεφαλαίου έχει παρουσιαστεί πέρα απο το απαραίτητο θεωρητικό υπόβαθρο και ένας οδηγός για το πως μπορούμε να το εγκαταστίσουμε και να το τρέξουμε.

Το τελευταίο κεφάλαιο είναι αφιερωμένο στην βιβλιογραφία και τις παραπομπές σε ηλεκτρονικές διευθύνσεις που έχουν γίνει.

1. Υποδομή Google

Η Google με σκοπό να μπορέσει να στηρίξει τις εφαρμογές της όπως είναι το Gmail, το Google Calendar κ.α. έφτιαξε ένα τεράστιο data center, με γύρω στους 20 κόμβους, το οποίο είναι κατανεμημένο σε όλο το κόσμο. Αποτελείται απο χιλιάδεις μηχανήματα και δίκτυα υψηλών ταχυτήτων για να διασυνδέουν τα data centers μεταξύ τους. Το παρακάτω σχήμα δέιχνει σε ποια σημεία σε όλο τον κόσμο έχει κόμβους το data center της Google και πως διασυνδέονται μεταξύ τους.



Σχήμα 1. Κόμβοι data center Google (Πηγή: «Using Google App Engine», *Charles Severance,* O'Reilly).

Όλοι αυτοί οι κομβοι είναι πολύ προσεκτικά κατασκευασμένοι ανάλογα με τον αριθμό χρηστών που έχουν να εξυπηρετήσουν αλλά και των υπηρεσιών που έχουν να εξυπηρετήσουν. Έτσι όσο μεγαλώνει η χρήση μια υπηρεσίας συνεχώς αναβαθμίζονται αυτοί οι κόμβοι αλλά και τα δίκτυα που διασυνδέουν αυτούς για να μπορούν να παρέχουν υψήλης ποιότητας υπηρεσίες.

Επειδή η αλλαγή/αναβάθμιση στον εξοπλισμό και στα δίκτυα της Google γίνεται με πολύ μεγάλο ρυθμό έχει φτάσει σε σημείο να μπορεί να κρίνει σε ποιο node του data center θα επενδύσει για να παρέχει περισσότερα resources, πράγμα αρκετά πολύπλοκο αν αναλογιστούμε απο πόσα σε πλήθος Nodes αποτελείται το data center τους. Μάλιστα για να να είμαστε και αρκετά ακριβείς αλλά και να δείξουμε το μέγεθος της πολυπλοκότητας ούτε οι προγραμματιστές της Google δεν μπορούν να εντοπίσουν τις αλλαγές που γίνονται στα nodes του data center.

Για αυτό το λόγο η Google έχει φτιάξει ένα software framework το οποίο αποκρύπτει τις λεπτομέρεις απο αυτόν που το χρησιμοποιεί και δεν του εμφανίζει για παράδειγμα με ποιο/ποιά data center μιλάει ή με ποιον server. Έτσι για παράδειγμα όταν θέλουμε

να δούμε ένα email απο ένα συγκεκριμένο άνθρωπο απλά λέμε την διεύθυνση του email του και το framework κάνει την δουλειά για εμάς.

2. Γενικά για το Google App engine

Το Google app engine είναι μια υπηρεσία web hosting για web applications στο cloud της Google. Λέγοντας web application εννοούμε αοποιαδήποτε εφαρμογή ή υπηρεσία στην οποία μπορούμε να έχουμε πρόσβαση απο έναν web browser (social network sites, mobile applications, storefronts κλπ.). Συγκεκριμένα το Google App Engine έχει σχεδιαστεί για να παρέχει scalable εφαρμογές σε πολλούς χρήστες ταυτόχρονα. Όσο περισσότεροι χρήστες χρησιμοποιούν την εφαρμογή το App engine θα πρέπει να παρέχει όλο και περισσότερους πόρους για να μπορέσουν να εξυπηρετηθούν οι χρήστες, χωρίς να χρειάζεται η εφαρμογή να γνωρίζει κάτι για αυτούς του πόρους, με άλλα λόγια ανάλογα με την επισκεψιμότητα κατανέμει και τους πόρους.

Σε αντίθεση με το κλασικό web hosting μοντέλο, με το Google App Engine, πληρώνεις μόνο για τους πόρους που χρησιμοποιείς. Αυτοί οι πόροι μετριούνται σε gigabytes χωρίς να χρειάζεται κάποια μηνιαία χρηματική συνδρομή. Συνδρομή υπάρχει μόνο για παραπάνω CPU usage, storage per month, incoming & outgoing bandwidth.

Αρχικά ο κάθε προγραμματιστής πέρνει κάποια resources δωρεάν, τα οποία είναι αρκετά για εφαρμογές μικρής επισκεψιμότητας. Η google με δικούς της υπολογισμούς υπολογίζει ότι με free resources μια μιρκή εφαρμογή μπορεί να φτάσει μέχρι και 5.000 επισκέψεις τον μήνα. Το App Engine αποτελείται απο τρία βασικά συστατικά τα οποία θα αναλυθούν περαιτέρω παρακάτω.

- **Runtime environment**
- **Datastore**
- **Scalable services**

2.1 Runtime Environment

Κάθε App Engine εφαρμογή αποκρίνεται σε web requests. Ένα web request ξεκινάει όταν ο client, μέσω ενός web browser, επισκέπτεται την εφαρμογή όπου ένα HTTP request συμβαίνει για να έρθει η σελίδα στον browser του. Όταν το App Engine λάβει αυτο το request βρίσκει την εφαρμογή που θέλει ο χρήστης απο το URL το οποίο έχει μορφή subdomain.appspot.com είτε απο το subdomain ενός custom domain το οποίο δηλώθηκε όταν έγινε η εγγραφή στην υπηρεσία. Το App Engine στην συνέχεια διαλέγει έναν server που να μπορεί να εξυπηρετήσει γρήγορα και αξιόπιστα, απο ένα πλήθος που έχει στην διάθεση της, καλεί την εφαρμογή την οποία δέχθηκε στο request και την στέλνει στον client.

Απο την πλευρά της εφαρμογής το runtime environment ξεκινάει να εκτελεί τις εργασίες του όταν του έρθει ένα request και σταματάει όταν τελειώνει. Το App engine χρησιμοποιεί τουλάχιστον δύο τρόπους για να αποθηκεύει κάπου κεντρικά τα requests που το έρχονται, αυτά όμως θα αναλυθούν αργότερα και δεν αποτελούν μέρος του runtime environment. Το Google App Engine μπορεί να κατανείμει την κίνηση σε παραπάνω απο έναν server για λόγους ταχύτητας και ποιότητας της εξυπηρέτησης των request που λαμβάνει.

Οι εφαρμογές που είναι στημένες στο Google App Engine δεν μπορούν να γράψουν στο filesystem του App Engine και προφανώς δεν μπορούν να διαβάσουν αρχεία που δεν τους ανήκουν. Μπορούν μόνο να διαβάσουν αρχεία τα οποία ανήκουν στην εφαρμογή και δίνουν το δικαίωμα να μπορεσει να τα διαβάσει κάποιος. Επίσης η εφαρμογή δεν μπορεί να έχει πρόσβαση σε hardware δικτυακούς πόρους του server και να τους παραμετροποιούν αλλά μπορούν να έχουν πρόσβαση σε υπηρεσίες που παραμετροποιούν συγκεκριμένα πράγματα.

Κάθε request πρέπει να επιστρέψει απάντηση το πολύ σε 30 δευτερόλεπτα. Αν και φαίνεται αρχικά μεγάλο το συγκεκριμένο χρονικό περιθώριο για web application το App Engine μπορεί να παραμετροποιηθεί για να εξυπηρετήσει ένα αίτημα σε λιγότερο απο 1 δευτερόλεπτο.

Το GAE προσφέρει δύο πιθανά runtime environments ένα για Java environment και ένα για Python environment ανάλογα με τη γλώσσα που επιθυμούμε να αναπτύξουμε την εφαρμογή που θέλουμε. Και τα δύο περιβάλλοντα χρησιμοποιούν το ίδιο application server μοντέλο: ένα request δρομολογείται στον application server, η εφαρμογή ξεκινάει στον application server και ακούει για requests για να μπορέσει να παράγει responses και να να τις δώσει στον application server να τις στείλει στον client.

Ενώ η χρήση διαφορετικών server για κάθε request έχει αποτέλεσμα να κερδίζουμε σε scaling στην εφαρμογή μας είνα αρκετά χρονοβόρο να σηκώνεται κάθε φορά ένα καινούριο instance για κάθε request που λαμβάνουμε. Το GAE όμως ξεπερνάει αυτο το πρόβλημα αποθηκεύοντας το application στην μνήμη του application server για όσο το δυνατόν περισσότερο και επαναχρησιμοποιώντας servers με έναν έξυπνο τρόπο. Όταν ένας server θέλει να ανακτήσει πόρους σβήνει την λιγότερο χρησιμοποιήμένη εφαρμογή που έχει στην μνήμη του. Όλοι οι application servers

έχουν το runtime environment προεγκατεστημένο πάνω τουςπριν φτάσει το πρώτο request. Όπότε μόνο η εφαρμογή χρειάζεται να φορτωθεί στον server.

Όσον αφαρά το λειτουργικό σύστημα και το hardware απο τα οποία αποτελείται το GAE χρησιμοποιείται linux μαζί με ένα cluster υπολογιστών. Αν και τελικά αυτό δεν έχει μεγάλη σημασία για runtime environment γιατί αυτό πάει και κάθεται πάνω απο το GAE και διαχειρίζεται τα resources της εφαρμογής, τις αιτήσεις εξυπηρέτησης κ.α. Δουλειές που χρειάζονται την χρήση του λειτουργικού συστήματος δεν θα συνατηθούν στο επίπεδο του runtime environment αλλά σε tasks που είναι εκτός αυτού.

2.2 Datastore

Το datastore αποτελέι μια βάση δεδομένων η οποία βασίζεται στο big table της Google. Έχει κατανεμηνμένη αρχιτεκτονική που επιτρέπει την αυτόματη διαχείριση μεγάλης ποσότητας δεδομένων. Το datastore θα μπορούσε καλύτερα να χαρακτηριστεί σαν object database μιας και δεν είναι μια παραδοσιακή relational database με joins, queries κλπ. Με την έννοια ότι παρέχει ότι ακριβώς και μια relational database αλλά με διαφορετική λογική. Εδώ θα συνατήσουμε objects τα οποία μπορούμε να κάνουμε commit στην βάση μας, Μοιάζει αρκετά με την αρχιτεκτονική του JPA (Java Persistency Architecture), όπου έχουμε entities και κάθε entity έχει κάποια properties. Στο πλαίσιο αυτό θεωρείται σκόπιμο να εξηγήσουμε κάποια πράγματα για entities και properties.

Το datastore έχει δύο εκδόσεις διαθέσιμες στο Google App Engine. Η μία έκδοση είναι η high replication datastore η οποία είναι και η default επιλογή στο GAE και η άλλη η master/slave datastore.

2.2.1 Master/slave datastore

Το master/slave replication σύστημα που μας παρέχει το datastore δίνει την δυνατότητα να γίνονται replicate τα δεδομένα μας ασύγχρονα σε άλλα data center κατα την διάρκεια που γράφουμε τα δεδομένα μας. Σε αυτό το μοντέλο κάθε φορά μόνο ένα datacenter είναι υπεύθυνο για να αποθηκεύει δεδομένα, το λεγόμενο master. Έτσι έχουμε μεγάλη συνοχή στα δεδομένα που ζητάνε τα queries. Το κακό σε αυτό το μοντέλο είναι ότι τα δεδομένα μας μπορεί να είναι μη διαθέσιμα σε περίπτωση που το σύστημα είναι κάτω για λόγους συντήρησης η απλά επειδή έχει πέσει.

2.2.2 High replication datastore

Το μοντέλο High Replication Datastore (HRD) είναι το default μοντέλο που υποστηρίζει το GAE όταν δημιουργούμε σε αυτό μια καινούρια εφαρμογή. Η διαθεσιμότητα των δεδομένων με αυτή τη μέθοδο είναι αρκετά μεγάλη ενω δεν

υπάρχει καμία πιθανότητα να χαθούν δεδομένα. Τα δεδομένα μας γίνονται replicate σε πολλά datacenters του Google χρησιμοποιώντας στο αλγόριθμο Paxos.

Αυτό το μοντέλο παρέχει την υψηλότερη διαθεσιμότητα για δίαβασμα και εγγραφή στην βάση δεδομένων και χρησιμοποιεί περίπου 3 φορές περισσότερο χώρο στην κατανεμημένη βάση και CPU απ' ότι μία master/slave datastore.

	High Replication	**Master/Slave**
Cost		
Storage	1x	1/3x
Put/Delete CPU	1x	5/8x
Get CPU	1x	1x
Query CPU	1x	1x
Performance		
Put/Delete Latency	1/2x–1x	1x
Get Latency	1x	1x
Query Latency	1x	1x
Consistency		
Get/Put/Delete	Strong	Strong
Most Queries	Eventual	Strong
Ancestor Queries	Strong	Strong
Occasional Planned Read-Only Period	No	Yes
Unplanned Downtime	Extremely rare. No data loss.	Rare. Possible to lose a small % of writes that occurred near the downtime (recoverable after event).

Συγκριτικός πίνακας High Replication Datastore/Master-Slave datastore.(Πηγή: http://code.google.com/intl/el-GR/appengine/docs/python/datastore/hr/)

2.2.3 Entities & Properties

Μία εφαρμογή που είναι στημένη στο GAE αποθηκεύει τα δεδομένα της σε κάποιο/κάποια entities του datastore. Κάθε ένα entity έχει και τουλάχιστον ένα property, καθένα οτ οποίο έχει μια ονομασία και ένα τύπο. Κάθε entity είναι συγκεκριμένου τύπου και αναπαριστά συγκεκριμένου τύπου δεδομένα.

Αρχικά αυτή η προσέγγιση φαίνεται αρκετά συναφής με τη παραδοσιακή relational database προσέγγιση. Τα entities ενός συγκεκριμένου τύπου είναι σαν rows σε έναν πίνακα και τα properties σαν fields του πίνακα. Όμως υπάρχουν οι εξής διαφορές ανάμεσα σε entities και rows. Δύο entities συγκεκριμένου τύπου δεν χρειάζεται να

έχουν τα ίδια properties και ακόμη κ αν έχουν ίδια properties δεν είναι απαραίτητο ότι θα είναι ίδιου τύπου οι τιμές τους. Έτσι καταλαβαίνουμε ότι τα datastore entities είναι "schemaless".Επίσης άλλη μια διαφορά είναι ότι κάθε entity έχει ένα μοναδικό κλειδί το οποίο είτε δημιουργείται απο την εφαρμογή η οποία μιλάει με την βάση είτε απο το ίδιο το GAE. Αυτό αποτελεί μια μεγάλη διαφορά σε σχέση με το παραδοσιακό relational database μοντέλο όπου το κλειδί ενός πίνακα είναι πεδίο (field) του πίνακα.Το κλειδί δεν μπορεί να αλλάξει ούτε τιμή ούτε τύπο απο την στιγμή που θα δημιουργηθεί και αυτό γιατί είναι το βασικό χαρακτηριστικό που ξεχωρίζει τα entities μεταξύ τους.

2.2.4 Queries & Indexes

Ένα query αποτελεί μια αναζήτηση η οποία επιστρέφει μια λίστα απο entities τα οποία ικανοποιούν τους κανόνες αναζήτησης που έχουμε θέσει ή δεν επιστρέφουν τίποτα αν δεν βρούν κάτι σύμφωνα με τους κανόνες. Ακόμη μπορούν να γυρίσουν πέρα απο ολόκληρα entities και τα keys μόνο των entities που αναζητούμε. Ενώ τέλος έχουν την δυνατότητα ταξινόμησης ως προ κάποιο χαρακτηριστικό τους των προς επιστροφή αντικειμένων.

Σε μια παραδοσιακή relational database τα queries σχεδιάζονται και εκτελούνται πάνω σε πίνακες real time ενώ ο προγραμματιστής μπορεί να θέσεις Indexes σε συγκεκριμένα πεδία του πίνακα για να έχει πιο γρήγορα αποτελέσματα κατα την εκτέλεση των queries. Απο την άλλη το datastore κάνει κάτι αρκετά διαφορετικό. Κάθε query έχει ένα μοναδικό Index που έχει λάβει απο το datastore. Έτσι όταν μια εφαρμογή θέλει να τρέξει ένα query το GAE πάει και βρίσκει το Index αυτού του query και επιστρέφει τα δεδομένα στην εφαρμογή. Βέβαια αυτό πουποθέτει ότι η εφαρμογή θα πρέπει να γνωρίζει εκ των προτέρων ποια queries θα εκτελέσει. Δεν χρειάζεται να γνωρίζει τις τιμές η τα φίλτρα εκ των προτέρων, αλλά χρειάζεται να γνωρίζει τον τύπο του entity που θα εκτελέσει το query, τα properties που θα φιλτραρουν ή θα ταξινομήσουν τα δεδομένα που θα μας επιστραφούν και την σειρά των ταξινομήσεων αν είναι πάνω απο μία.

Το App Engine παρέχει ένα σύνολο απο indexes για απλά queries το οποίο βασίζεται στα properties που υπάρχουν στον τύπο του entity που θα εκτελεστεί το query. Για πιο πολύπλοκα queries θα πρέπει να ορίζονται οι κανόνες δημιουργίας Indexes κατα το αρχικό configuration της εφαρμογής. Το GAE βοηθάει στην δημιουργία αυτού του αρχικού configuration βρίσκοντας ποια query εκτελούνται καθώς τεστάρεται η εφαρμογή τοπικά, πριν την σηκώσουμε στο cloud της Google. Έτσι όταν θα φορτωθεί η εφαρμογή στο Google το datastore θα γνωρίζει τα Indexes κάθε query.

Όταν η εφαρμογή μας δημιουργεί καινούρια entities ή τροποποιεί κάποιο ήδη υπάρχον, το datastore τροποποιεί και το αντίστοιχο Index. Αυτό κάνει την εκτέλεση των queries πολύ γρήγορη μιας και κάθε query είναι ένα μια απλή σαρωση στον πίνακα. Στην πραγματικότητα ένα query στο datastore δεν επηρεάζεται απο το πληθος των entities που έχουμε στο datastore παρα μόνο απο το μέγεθος των δεδομένων που μας επιστρέφουν.

2.2.5 Transactions

Όταν σε μια εφαρμογή έχουν συνδεθεί πολλοί clients οι οποίοι προσπαθούν να διαβάσουν και να γράψουν στα ίδια δεδομένα ταυτόχρονα είναι υποχρεωτικό τα δεδομένα να έχουν συνοχή (consistent). Ο χρήστης δεν πρέπει ποτέ να βλέπει ημιτελή δεδομένα επειδή καποιος άλλος χρήστης δεν πρόλαβε να τελειώσει την δουλειά που κάνει. Όταν μια εφαρμογή κάνει update στα properties ένος entity, το App Engine μας διασφαλίζει είτε ότι όποιο update γινόταν στην εφαρμογή κατάφερε να πραγματοποιηθεί είτε απέτυχε οπότε δεν γίνεται κανένα update στο entity και μένει όπως ήταν ακριβώς πριν την προσπάθεια να τροποιποιηθεί. Έτσι οι άλλοι χρήστες δεν βλέπουν καμία αλλαγή μέχρι να ολοκληρωθεί η διαδικασία του update. Το update ενός entity είναι ένα transaction με το datastore. Κάθε transaction μπορεί είτε να πετύχει ολοκληρωτικά είτε να αποτύχει ολοκληρωτικά.

Σε κάθε transaction μπορούν να διαβαστούν και να γίνουν update πολλά entities με τον μόνο περιορισμό ότι κατα την δημιουργία των entities θα πρέπει να οριστούν στο GAE τα entities οι συσχετίσεις των entities και ποια θα πρέπει να τροποποιούνται μαζί. Αυτό επιτυγχάνεται απο την εφαρμογή δημιουργώντας entities στο entity group. Το GAE χρησιμοποιεί αυτή την πληροφορία για να δει πως είναι κατανεμημένα τα entities στους server προκειμένουν να εγγυηθεί ότι ένα transaction πέτυχε η απέτυχε ολοκληρωτικά.

Όταν μια εφαρμογή χρησιμοποιεί το datastore API για να τροποποιήσει ένα entity, ο έλεγχος δεν επιστρέφει στην εφαρμογή παρα μόνο όταν το state του transaction γίνει successful ή fail. Μετά ο έλεγχος μπορεί να επιστρέψει στην εφαρμογή και να την ενημερώσει ότι το transaction πέτυχε ή απέτυχε ολοκληρωτικά. Για την περίπτωση του update ενός entity αυτό σημαίνει ότι η εφαρμογή μας περιμένει όλα τα entities και τα indexes να δημιουργηθούν πριν προχωρήσει σε άλλα tasks. Εαν δύο χρήστες προσπαθούν να τροποποιήσουν ένα entity ταυτόχρονα το datastore αποκρίνεται και στους δύο με ένα failure exception. Το GAE χρησιμοποιεί optimistic concurrency control.

Το διάβασμα ενός entity δεν αποτυγχάνει ποτέ χάρη στο optimistic concurrency. Η εφαρμογή απλά διαβάζει την πιο πρόσφατη κατάσταση του entity. Επίσης υπάρχει η δυνατότητα να διαβαστεί πολλές φορές ένα entity μέσα σε ένα transaction για να διαπιστωθεί ότι τα δεδομένα που διαβάστηκαν έχουν συνοχή και είναι τα πιο πρόσφατα. Στις περισσότερς των περιπτώσεων να ξαναδοκίμασουμε το ίδιο transaction πάνω στο ίδιο entity θα εκτελεστεί με επιτυχία. Όμως αν η εφαρμογή έχει τέτοιο σχεδιασμό που πολλοί χρήστες να προσπαθούν ταυτόχρονα να τροποποίησουν το ίδιο entity το transaction τους δεν θα μπορέσει να ολοκληρωθεί με επιτυχία λόγω concurrency failure. Επομένως είναι σημαντικό τα entity groups να σχεδιαστούν έτσι ώστε να αποφεύγονται concurrency failures σε πειπτώσεις που την εφαρμογή αμς την χρησιμοποιούν πολλοί users ταυτόχρονα.

Σε κάθε εφαρμογή μπορούμε να έχουμε πολλές δουλειές να γινονται σε κάθε ένα transaction. Για παράδειγμα όταν ξεκινάει η εφαρμογή μας μπορεί να ξεκινήσει ένα transaction το οποίο θα διαβάζει ένα entity, θα κάνει ένα update σε κάποιο property

του και στην συνέχει θα σώζει την αλλαγή. Σε αυτή την περίπτωση η εφαρμογή δεν η αποθήκευση των αλλαγών δεν συμβαίνει εαν ολόκληρο το transaction δεν τελειώσει με επιτυχία. Αν κάτι γίνει και αποτύχει η εφαρμογή θα πρέπει να δοκιμάσει να τρέξει απο την αρχή πάλι ολόκληρο το transaction.

Συνοψίζοντας θα μπορούσαμε να πούμε οτι το GAE με την χρήση των Indexes και του μηχανισμού optimistic concurrency δίνει την δυνατότητα δημιουργίας εφαρμογών οι οποίες διαβάζουν γρήγορα και έγκυρα δεδομένα ακόμη σε εφαρμογές με πολλούς χρήστες.

2.3 Services

Τα services αποτελούν τη συσχέτιση ανάμεσα στο datastore και το runtime environment. Η εφαρμογή χρησιμοποιεί ένα API για να μπορέσει να χρησιμοποιήσει ένα ξεχωριστό σύστημα το οποίο διαχειρίζεται τις δικές του ανάγκες για scaling ξεχωριστά απο το runtime environment.

Το memory cache (memcahe) service είναι ένα μικρό σε χρονική διάρκεια service το οποίο αποθηκεύει τα key values και έχει σαν βασικό πλεονέκτημα ότι ειναι πολύ γρήγορα πολύ πιο γρήγορα εαν θα χρησιμοποιούσαμε το datastore για να αποθηκέυσουμε ή να διαβάσουμε. Το memcache service αποθηκεύει δεδομένα στην μνήμη και όχι στον σκληρό δίσκο για να έχει πιο γρήγορη προσπέλαση σε αυτά. Έχει κατανεμημημένη αρχιτεκτονική όπως και το datastore αλλά δεν είναι persistent με την έννοια ότι αν πέσει ο server κατα την διάρκεια που το χρησιμοποιούμε η μνήμη διαγράφεται. Όπως υποδηλώνεται και το το όνομα αυτού του service είναι καλύτερο να χρησιμοποιείται σαν μία προσωρινή μνήμη cache για τα αποτελέσματα queries η υπολογισμών οι οποίοι επαναλαμβάνονται με μεγάλη συχνότητα. Λειτουργεί ως εξής, ψάχνει για μια cached value μέσα στην μνήμη, αν δεν την βρεί τότε εκτελεί το query ή τον υπολογισμό και τα αποθηκεύει στην cache για μελλοντική χρήση.

Οι εφαρμογές που είναι στο GAE έχουν την δυνατότητα να έχουν πρόσβαση και σε άλλα web resources χρησιμοποιώντας το URL Fetch service. Με αυτό το service η εφαρμογή μας στέλνει HTTP requests σε άλλους servers στο Internet για να χρησιμοποιήσει web services ή ακόμα και να φέρει ολόκληρες σελίδες. Επειδή οι απομαρκυσμένοι server Μπορεί να είναι αργοί στην απόκριση τους το URL Fetch service τρέχει το fetching στο background ενώ ένας request handler κάνει άλλα πράγματα ταυτόχρονα. Προσοχή όμως το fetch πρέπει να εκτελεστεί μέσα διάστημα που είναι ενεργός ο request handler. Επίσης υπάρχει ένα χρονικό περιθώριο μέσα στο οποίο θα πρέπει να αποκριθεί ο απομακρυσμένος server, το οποίο αν ξεπεραστεί σταματάει η προσπάθεια επικοινωνίας μαζί του.

2.4 Google Accounts

Το GAE δίνει την δυνατότητα Integration με Google accounts σε gmail, Google docs και Google calendar. Μπορούμε να χρησιμοποιήσουμε αυτά τα accounts στην εφαρμογή μας και αν ο χρήστης έχει ήδη να μπορεί να κάνει αυτόματα Login και στις παραπάνω υπηρεσίες του Google αυτόματα. Έτσι μπορούμε ενα θέλουμε, χωρίς αυτό να είναι υποχρεωτικό, να βάλουμε τον χρήστη να χρησιμοποιήσει το Google account και να μην χρειαστεί τον να βάλουμε να φτιάξει καινούριο για το δικό μας σύστημα.

Τα Google accounts είναι ιδιαίτερα χρήσιμα να χρησιμοποιούνται όταν φτιάχνουμε εφαρμογές για επιχειρήσεις και οργανισμούς οι οποίοι ήδη χρησιμοποιούν google applications. Έτσι οι χρήστες της εφαρμογής θα χρησιμοποιούν ένα κοινό password για να μπορέσουν να κάνουν sign in και στην εφαρμογή μας και στις εφαρμογές του Google.

2.5 Task queues & cron jobs

Μια web εφαρμογή πρέπει να αποκρίνεται σε web requests πολύ γρήγορα, συνήθως σε λιγότερο απο ένα δευτερόλεπτο. Πράγμα το οποίο σημαίνει ότι η εφαρμογή έχει αρκετά περιορισμένο χρόνο να κάνει τους υπολογισμούς της. Γι αυτό το λόγο το GAE χρησιμοποιεί task queues τα οποία δίνουν την δυνατότητα σε request handlers να ορίσουν το task που έχουν να κάνουν αργότερα. Με τις ουρές διασφαλίζεται ότι κάθε task θα εκτελεστεί και ακόμη και σε πίπτωση που αποτύχει να εκτελεστεί η ουρά το εκτελεί ξανά μέχρι να εκτελεστεί επιτυχώς.Μία ουρά μπορεί να εκτελέσει ένα task καλώντας τον αντίστοιχο request handler. Απο την στιγμή που θα κληθεί ο handler έχει στην διάθεση του τριάντα δευτερόλεπτα να χρόνο να εκτελέσει την εργασία του.

Ένα ιδιαίτερα χρήσιμο και δυνατό προσόν του task queue είναι η δυνατότητα που έχει να φτιάχνει ουρές με datastore transactions. Αυτό μας διασφαλίζει ότι μόνο αν ένα task θα εκτελεστεί με επιτυχία θα περάσουμε στην εκτέλεση του επομενου στην ουρά. Έτσι μπορούμε να σπάσουμε ένα μεγάλο transaction σε μικρότερα κομμάτια και να τα βάλουμε να εκτελούνται το ένα μετα το άλλο σαν tasks στην ουρά.

Ακόμη το GAE έχει άλλο ένα service για την εκτέλεση tasks σε συγκεκριμένα χρονικά διαστήματα μέσα στην μέρα. Είναι scheduled tasks τα οποία βασίζονται σε ένα σύστημα που ήδη υπάρχει εδω και χρόνια στον linux το οποίο ονομάζεται cron. Τα cron tasks μπορούν να καλέσουν έναν handler μια συγκεκριμένη χρονική στιγμή (ώρα, μέρα, μήνας, έτος) και να κάνουν μια εργασία.

2.6 Developer tools

Το Google παρέχει εργαλεία για ανάπτυξη εφαρμογών για το App Engine σε Java και σε Python. Κάθε SDK αποτελείται απο έναν web server ο οποίος τρέχει την

εφαρμογή μας τοπικά και εξομοιώνει το running environment, το datastore και τα services.

Το development version του datastore μπορεί αυτόματα να δημιουργήσει configuration για Indexes κατα την διάρκει που η εφαρμογή εκτελεί queries. Έτσι το GAE μπορεί να χτίσει indexes για αυτά τα queries. Ακόμα το development version του web server περιλαμβάνει έναν built-in web application για να μπορεί να επικοινωνεί με το simulated datastore που έχουμε τοπικά.

Τέλος κάθε SDK έχει ένα tool για την επικοινωνία με το Google App Engine μέσω του οποίου μπορούμε να κάνουμε deploy την εφαρμογή μας στο GAE. Επίσης μπορούμε να χρησιμοποιήσουμε αυτό το tool για να κατεβάσουμε τα logs για την εφαρμογή μας στο μηχάνημα μας η ακόμη και να διαχειριστούμε τα Indexes της Live εφαρμογής μας.

2.7 Administration console

Ότα η εφαρμογή μας είναι έτοιμη για να γίνει deploy στο GAE, πρέπει πρώτα να φιάξουμε ένα account στο administration console του GAE. Αυτό το εργαλείο μας δίνει την δυνατότητα να δημιουργήσουμε και να διαχειριστούμε την εφαρμογή που θέλουμε να κάνουμε deploy, να δούμε την επισκεψιμότητα, logs, στατιστικά κλπ. Όλα αυτά αποτελούν το το administration console το οποίο είναι ένα web application που μας δείχνει όλα τα παραπάνω για την εφαρμογή μας.

Το συγκεκριμένο εργαλείο δίνει την δυνατότητα real-time performance της εφαρμογής και των δεδομένων που χρησιμοποιεί η εφαρμογή και θώς και πρόσβαση στα logs της εφαρμογής. Επίσης μπορούμε να τρέξουμε queries μέσω του web interface του administration console και να δούμε την κατάσταση των indexes του datastore.

Όταν κάνουμε upload καινούριο κώδικα στην εφαρμογή μας ανεβάζουμε και μια καινούρια Version της εφαρμογής μας. Η live όμως εφαρμογή αμς είναι αυτή που είναι μαρκαρισμένη σαν default. Ο προγραμματιστής ορίζει ποια έκδοση θα είναι η default. Εαν θέλουμε να δούμε τις non-default version για κάθε μια απο αυτές έχουμε ένα URL το οποίο όταν το χτυπήσουμε μπορούμε να δούμε στον web browser το το version της εφαρμογής που θέλουμε. Αυτό μας δίνει την δυνατότητα να τεστάρουμε Μια εφαρμογή πριν την κάνουμε τη βασική μας.

2.8 Πως τρέχουν οι εφαρμογές στο Google App Engine

Ο καλύτερος τρόπος για να παραλληλίσουμε το πως τρέχουν οι εφαρμογές μας στο cloud της Google είναι ότι όλα είναι virtual. Δεν υπάρχει τίποτα physical που να μπορεί να έρθει σε άμεση επαφή ο προγρμματιστής που κάνει deploy την εφαρμογή

του στο Google App Engine. Είναι σας να έχουμε ένα μάυτο κουτί το οποίο χωρίς να ξέρουμε πως και τι μας προσφέρει μια υψηλού επιπέδου υπηρεσία με πολύ μεγάλη διαθεσιμότητα.

Σε ένα πιο παραδοσιακό περιβάλλον hosting web εφαρμογών οι server είναι γνωστοί και έχουν σταστική IP και τοποθεσία. Επομένως όπως φαίνεται είναι γνωστές κάποιες πληροφοριές για το μηχάνημα που κάνει hosting την εφαρμογή. Απο την άλλη το cloud της Google μοιάζει πολύ στην δομή του με τα δίκτυα κινητής τηλεφωνίας. Τα προγράμματα και δεδομένα βρίσκονται κάπου στο cloud χωρίς να μας ενδιαφέρει η τοποθεσία τους και με κάποιο τρόπο βρίσκουν τον δρόμο για να επικοινηνωνήσουν και να ανακτηθούν αντίστοιχα απο την εφαρμογή που έχει κάποιο αίτημα γι' αυτά. Έτσι για παράδειγμα αν έχουμε μια εφαρμογή στο παρακάτω domain *myApplication.appspot.com* το Google μπορεί να δώσει στην εφαρμογή που είναι στημένη εκεί μια διαφορετική IP ανάλογα με την τοποθεσία του server στο κόσμο. Έτσι μπορούμε να πούμε ότι η εφαρμογή μας δεν ξέρουμε σε ποιο server της Google θα τρέχει κάθε φορά, αυτό μπορεί να αλλάζει. Όμως κάθε φορά η Google φροντίζει να ενημερώνει τους DNS server της έτσι ώστε να έχει κάθε φορά η διεύθυνση μας (*myApplication.appspot.com)* την αντίστοιχη IP του server στον οποίο φιλοξενείται. Επίσης όταν πάρει IP η εφαρμογή μας τότε η Google η ίδια είναι υπεύθυνη να δει με ποιο data center μπορεί να επικοινωνήσει. Η επιλογή στηρίζεται στην τοποθεσία του server και του data center, όσο πιο κοντα σε απόσταση γίνεται. Στην περίπτωση βεβαία που η εφαρμογή είναι αρκετά δημοφιλής, δηλαδή έχουμε αλλάξει το συμβόλαιο απο free σε Commercial, η εφαρμογή μας μπορεί να τρέχει ταυτόχρονα σε πολλά data center. Όταν η εφαρμογή δέχεται πολλά requests μπορεί να σηκώθει και ενα διαφορετικό Instance της εφαρμογής κάποιο διαφορετικό server σε διαφορετική τοποθεσία, έτσι μπορεί ταυτόχρονα να τρέχει η εφαρμογή σε πολλούς server ανα τον κόσμο. Όλη αυτή η πληροφορία για το που τρέχει η εφαρμογή μας και με ποιο data center επικοινωνεί βλέπουμε ότι διαμορφώνεται δυναμικά στον χρόνο πράγμα που συμαίνει ότι είναι πολύ δύσκολο να ενημερώνόμαστε και στις περισσότερες περιπτώσεις δεν μας ενδιαφέρει κιόλας. Η Google με εσωτερικούς μηχανισμούς είναι υπεύθυνη να στέλνει αιτήματα που έρχονται απο έναν χρήστη προς την αντίστοιχη εφαρμογή στην οποία απευθύνονται.

Συνομίζοντας καταλαβαίνουμε ότι το να τρέχει μια εφαρμογή στο cloud της Google είναι μάλλον μια αρκετά πολύπλοκη διαδικασία η οποία αποκρίπτεται απο τον χρήστη ως προς τις επιμέρους λειτουργίες της. Μπορούμε να το σκεφτούμε σαν ένα μία πρήζα στην οποία εμείς συνδέουμε την εφαρμογή μας και απλά παίζει χωρίς να γνωρίζουμε εμείς την υποδομή που υπάρχει απο μέσα, άλλωστε αυτή είναι και η βασική ιδέα του cloud computing.

2.9 Γιατί το Google App Engine

Αρχικά θα σκεφτόταν κανείς γιατί να ανεβάσουμε την εφαρμογή μας στο cloud της Google. Γιατί να μην έχουμε μόνοι μας το έλεγχο της εφαρμογής και να μην περνουμε έμεις ότι απόφαση θέλουμε για το μέλλον της εφαρμογής. Αρχικά θα μπορούσαμε να πούμε ότι το κόστος για να αγοράσεις όλα αυτα τα μηχανήματα και να τα στήσεις και να τα διαχειριστείς είναι μεγάλο. Και επειδή ζούμε και σε μια «πράσινη» εποχή όπου όλο και περισσότερο σκεφτόμαστε το περιβάλλον να μην ξεχνάμε πόση ενέργεια καταναλώνουν όλα αυτα.

Επίσης θα μπορούσαμε να πούμε ότι το Google App Engine σε γλιτώνει απο ένα σωρό ερωτήματα που δημιουργούνται όταν θέλουμε να στήσουμε ένα μια εφαρμογή στο διαδίκτυο. Για παράδειγμα τι λειτουργικό σύστημα θα τρέχει ο server πάνω στον οποίο θα στήσουμε την εφαρμογή, ποιά είναι η πιο αξιόπιστη και στα μέτρα μου version του λειτουργικού συστήματος, πως θα ελέγξω ότι δεν θα υπάρξει εισβολή στον server, χρειάζομαι firewall, χρειάζομαι cluster και πολλές ακόμα σημαντικές ερωτήσεις. Με το Google App Engine όλα αυτά τα ερωτήματα απαντούνται και μάλιστα ελαχιστοποιούνται οι πιθανότητες να υπάρξει κάποιο πρόβλημα. Φυσικά γνωρίζοντας το κύρος μια τόσο μεγάλης εταιρείας και γνωρίζοντας την ποιότητα και την ασφάλεια των υπηρεσιών που προσφέρει η χρήση του Google App Engine θεωρείται μια αρκετά αξιόπιστη λύση.

2.10 Πλεονεκτήματα Google App Engine

Το Google App Engine είναι ένα παίσχυρο PaaS (Platform as a Service) γιατί μας παρέχει ένα στημένο σύστημα για να μπορέσουμε απλά να κάνουμε deploy την εφαρμογή. Έτσι δεν είναι αναγκαίο να στήσουμε και να καταβάλουμε και αρκετό κόπο για να συντηρήσουμε τον server στον οποίο είναι στημένη η εφαρμογή.

Επίσης μας παρέχει δύο γλώσσες στις οποίες μπορεί να είναι γραμμένες οι εφαρμογές μας (Java/Python) και παρέχει στον προγραμματιστή ένα φιλικό περιβάλλον για deployment στον server ακόμη και μέσα απο το IDE (Eclipse plugin).

Άλλο ένα πολύ μεγάλο πλεονέκτημα είναι το administration console το οποίο μας δίνει την δυνατότητα να διαχειριζόμαστε της εφαρμογή και να βλέπουμε διάφορα χρήσιμα στοιχεία για την επισκεψιμότητα, την βάση δεδομένων, τα Logs Κλπ.

Τέλος το Google App Engine μπορεί να χρησιμοποιηθεί δωρεάν μέχρι κάποιο όριο επισκεψιμότητας (bandwidth) και μεγέθους βάσης δεδομένων. Αυτό αυτομάτως το καθιστά ένα πολύ ανταγωνιστικό PaaS γιατί μπορείς πολύ απλές εφαρμογές να τις ανεβάσεις στο web εντελώς δωρεάν.

2.11 Μειονεκτήματα-Περιορισμοί Google App Engine

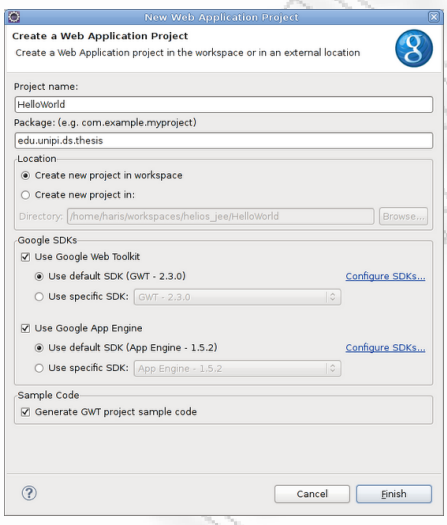
Πέρα απο τα πολλά πλεονεκτήματα όμως υπάρχουν και κάποια μειονεκτήματα ή μάλλον περιορισμούς που εισάγει το Google App Engine στην free έκδοση του. Αρχικά μπορούν να υπάρχουν μόνο πέντε εκατομμύρια page views κάθε μήνα. Επίσης η απόκριση του server σε ενα request μπορεί να φτάσει μέχρι και τα 30 δευτερόλεπτα χρόνος αρκετά μεγάλος για web application. Το επισημαίνουμε όμως και πάλι πως όλα τα μειονέκτηματα και οι περιορισμοί παρουσιάζονται στην free έκδοση του Google App Engine.

2.12 Development & deployment στο Google App Engine

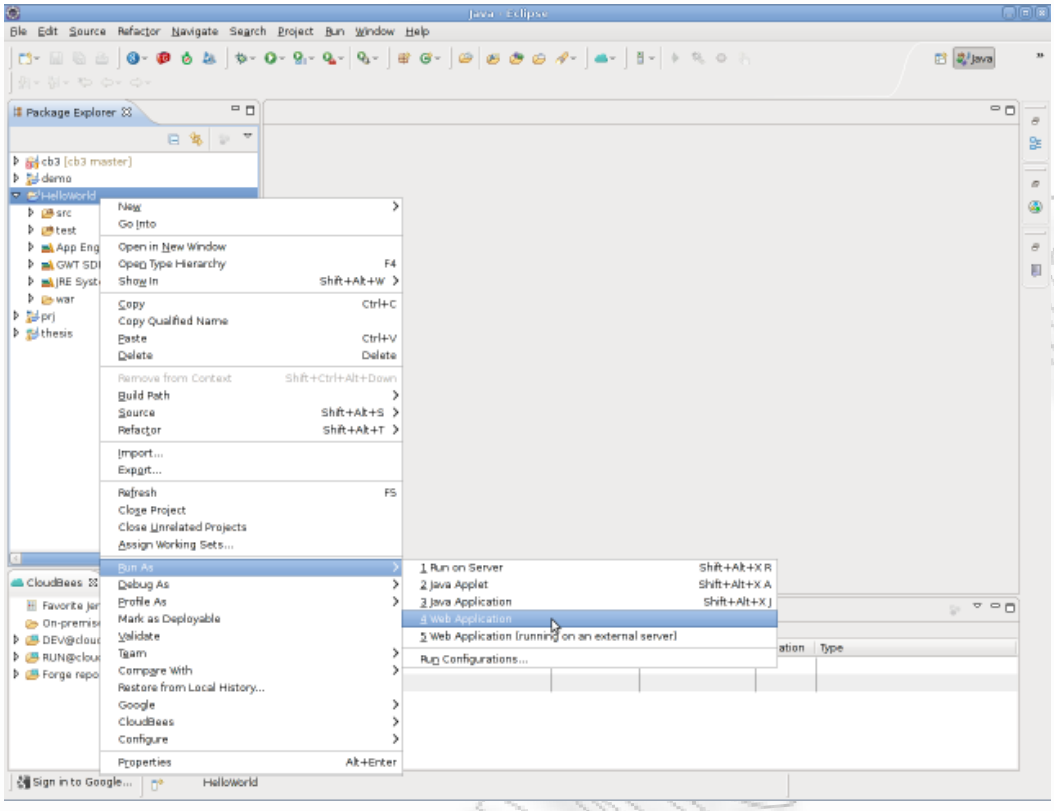
Η Google παρέχει ένα πρόσθετο για το περιβάλλον ανάπτυξης eclipse (http://www.eclipse.com). Με το πρόσθετο αυτό μπορούμε να αναπτύξουμε τοπικά την εφαρμογή μας, να την εκτελέσουμε τοπικά σε ένα περιβάλλον που προσομοιώνει το περιβάλλον εκτέλεσης του Google App Engine και να την κάνουμε deploy στην εν λόγω πλατφόρμα.

Το πρόσθετο είναι διαθέσιμο από τις σελίδες του Google App Engine ανάλογα με την έκδοση του eclipse. Για την έκδοση helios (3.6) η διεύθυνση είναι http://dl.google.com/eclipse/plugin/3.6

Αφού εγκαταστήσουμε το πρόσθετο, επανεκκινούμε το eclipse και πάμε να δημιουργήσουμε ένα νέο Web Application Project. Συμπληρώνουμε τα στοιχεία για την εφαρμογή και πατάμε Finish.

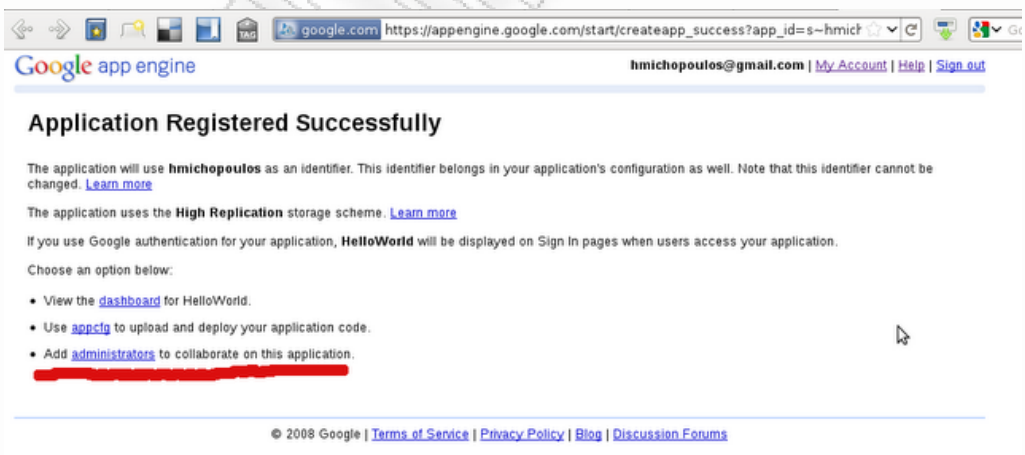


Έπειτα, "τρέχουμε" την εφαρμογή κάνοντας επάνω της δεξί κλικ->Run as->Web application

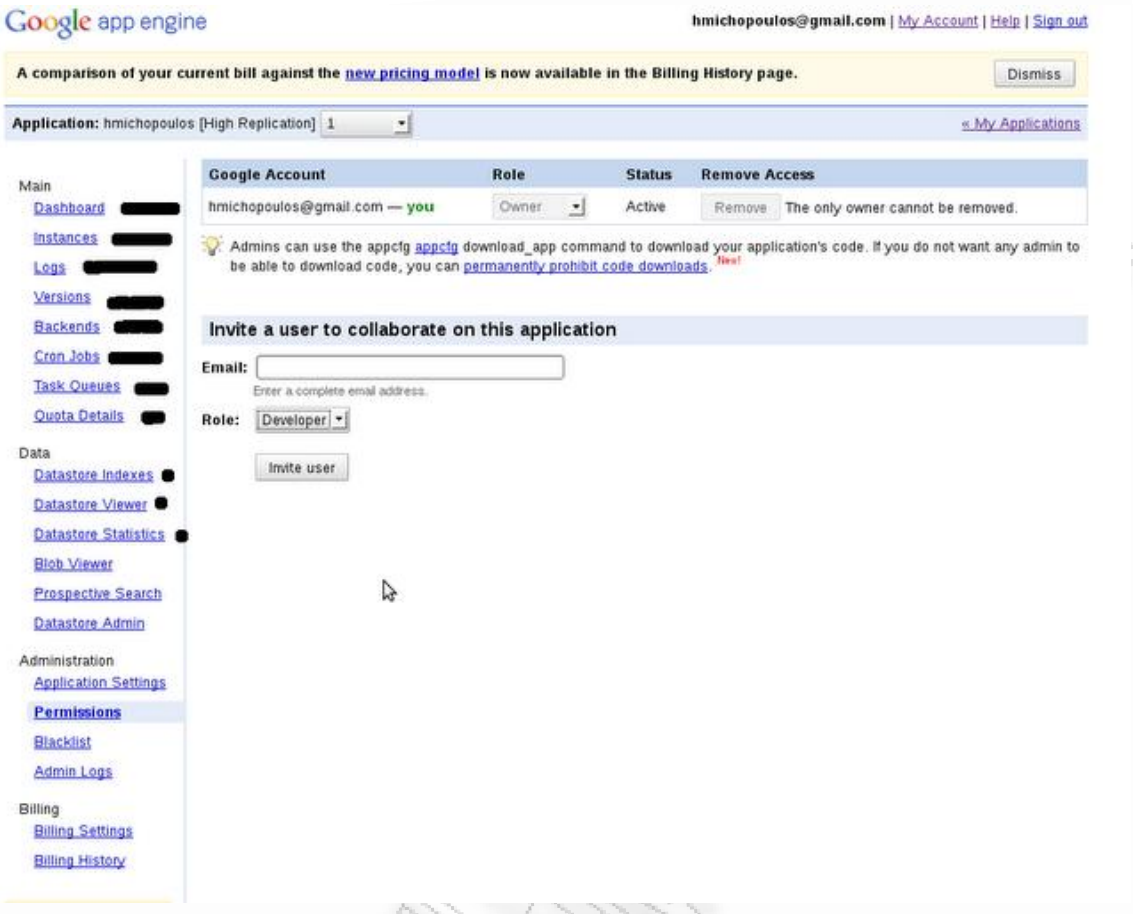


Στο παράθυρο που θα ανοίξει μπορούμε να δοκιμάσουμε την εφαρμογή μας. Εάν όλα είναι σωστά μπορούμε να την ανεβάσουμε στο Google App Engine. Για να ολοκληρώσουμε τη διαδικασία θα πρέπει να προσθέσουμε μια εφαρμογή στο https://appengine.google.com/start/createapp με το επιθυμητό Application ID.

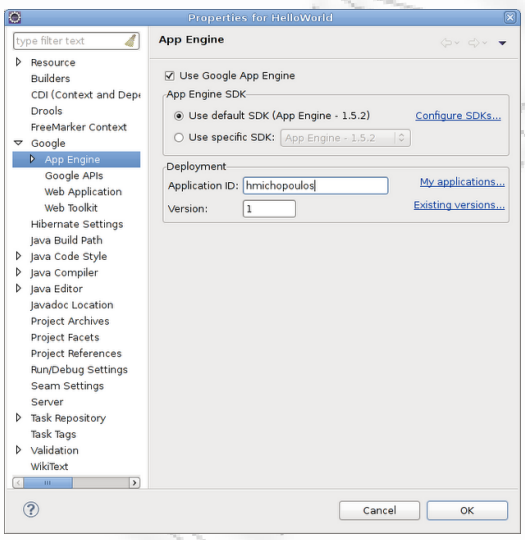
Μόλις ολοκληρωθεί η διαδικασία επιτυχώς, η σελίδα που θα εμφανιστεί θα μας ενημερώσει για την επιτυχία και θα μας προτρέψει να προσθέσουμε administrators και χρήστες στην εφαρμογή.



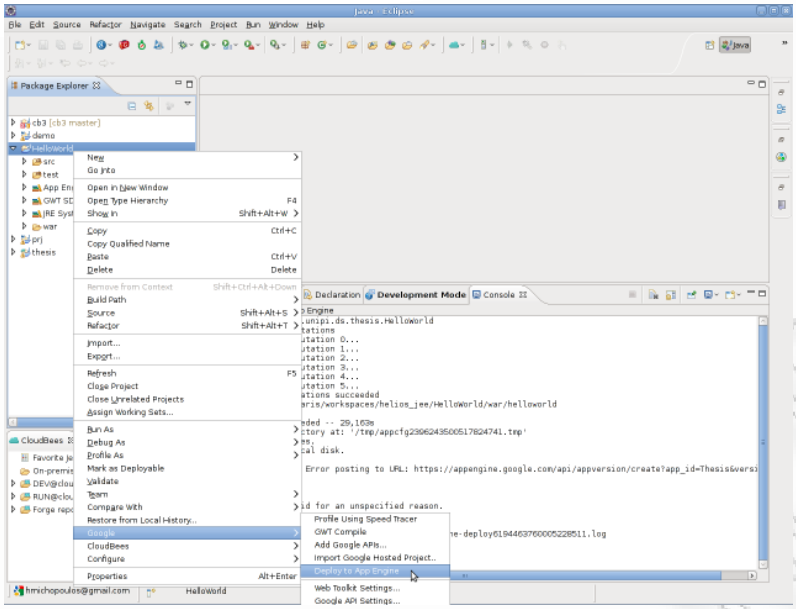
Στη σελίδα αυτή, εκτός από το User Account, μπορούμε να δούμε πληροφορίες για όλες τις υπηρεσίες του Google App Engine σχετικά με την εφαρμογή μας.



Έπειτα, πρέπει στο eclipse να δηλώσουμε αυτό το Application ID κάνοντας στην εφαρμογή μας δεξί κλικ->Properties->Google->App Engine.



Αφού ολοκληρώσουμε τα βήματα αυτά επιτυχώς μπορούμε να ανεβάσουμε την εφαρμογή μας στο Google App Engine κάνοντας δεξί κλικ στην εφαρμογή->Google->Deploy to App Engine

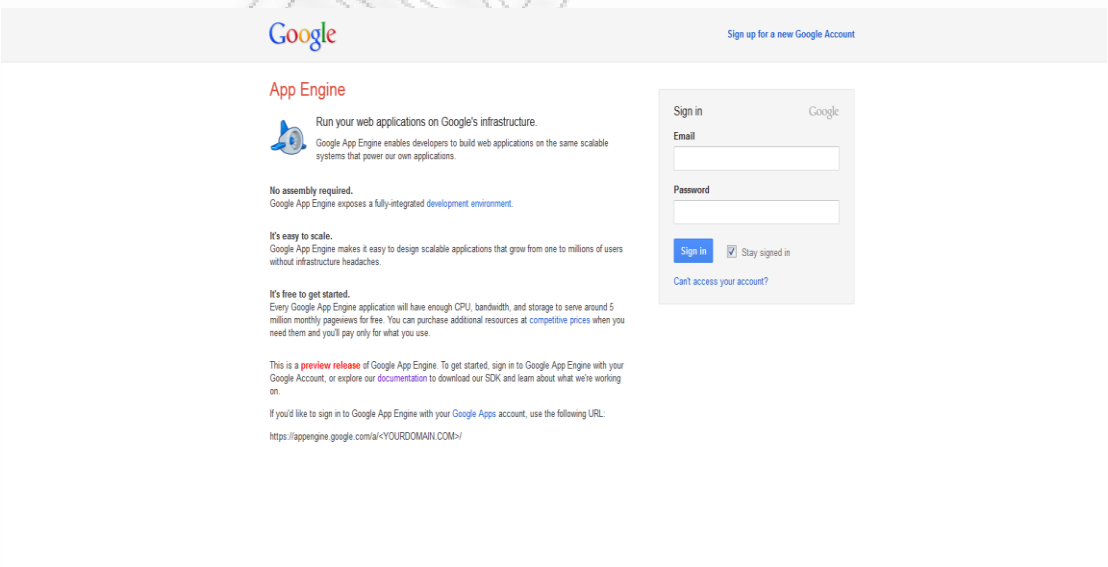


Η εφαρμογή μας θα είναι διαθέσιμη στη διεύθυνση http://applicationId.appspot.com, στο παράδειγμα http://spkpxl.appspot.com (να σημειωθεί ότι το συγκεκριμένο link δουλεύει).

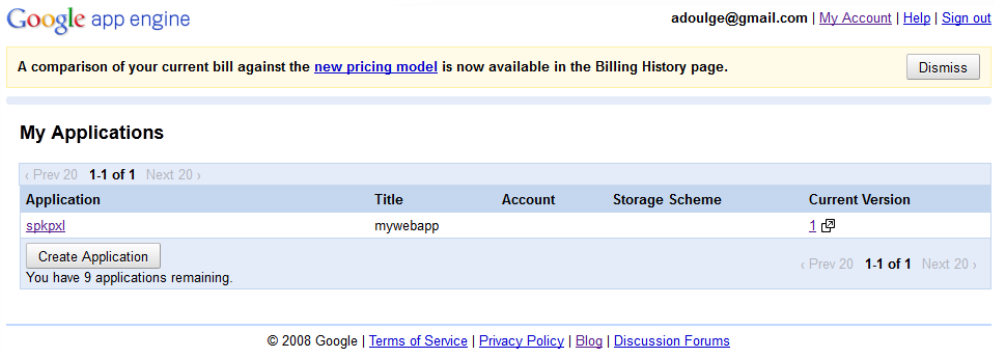
2.13 Monitoring στο Google App Engine στην πράξη

Μια απο τις πολύ μεγάλες δυνατότητες που μας δίνει το GAE είναι η δυνατότητα να διαχειριστούμε στην εφαρμογή που έχουμε κάνει deploy.

Αρχικά χτυπάμε το URL του GAE σε web browser (https://appengine.google.com/) και βλέπουμε την παρακάτω οθόνη για να κάνουμε Login να διαχειριστούμε την εφαρμογή μας.

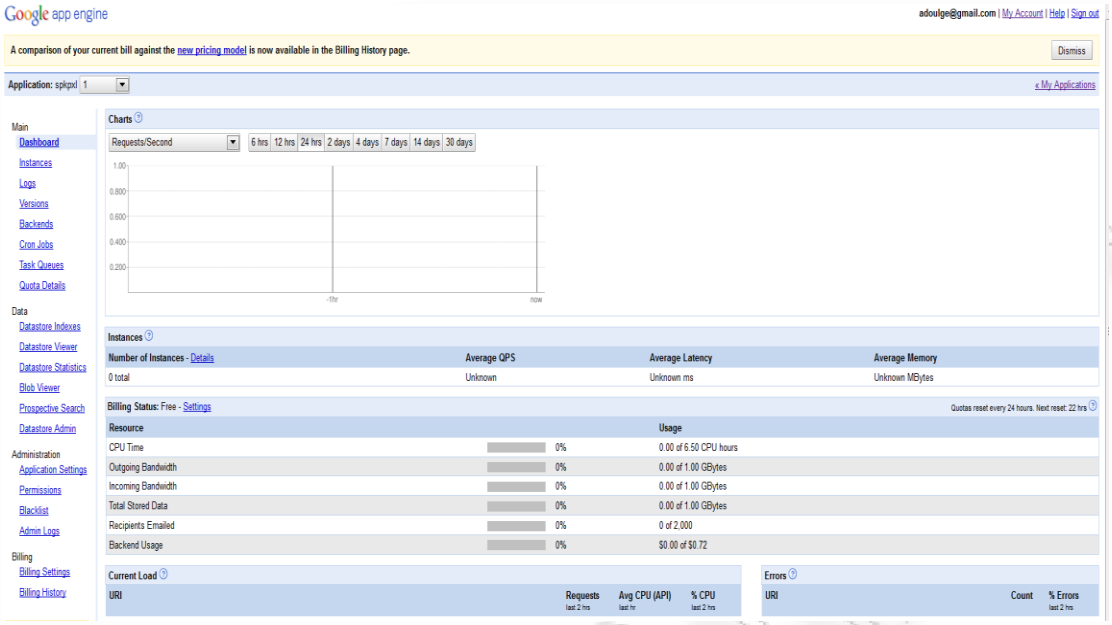


Βάζουμε το email και το password που ορίσαμε για το account μας κατα το sign up και μπαίνουμε στο σύστημα.

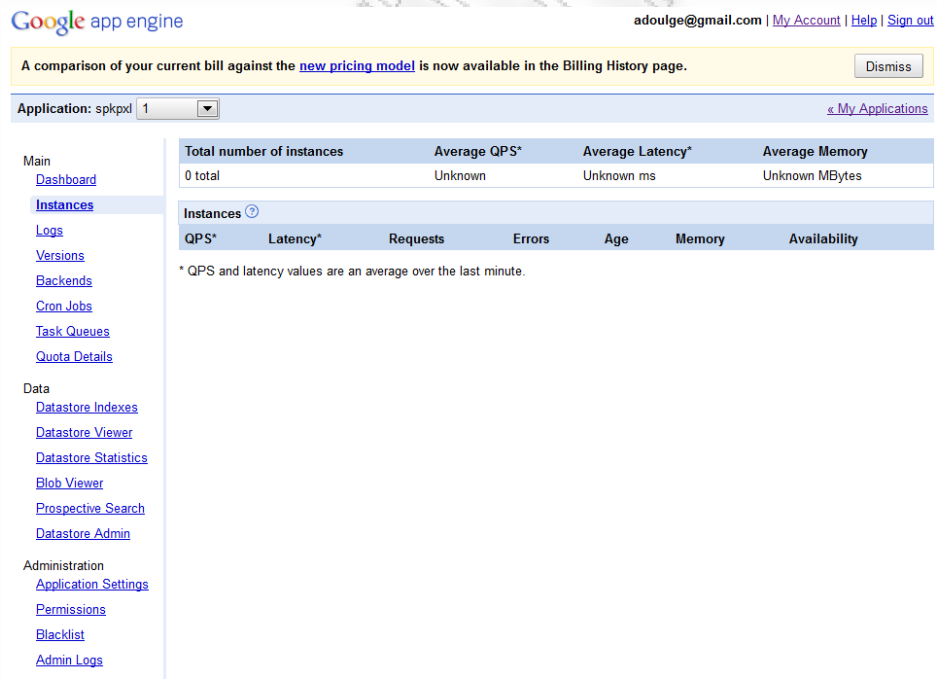


Στην αρχική σελίδα βλέπουμε τις εφαρμογές που έχουμε σηκώσει στο Google App Engine και επιλέγουμε αυτή που θέλουμε να διαχειριστούμε. Βλέπουμε ότι δεξιά κρατάει και version της εφαρμογής που έχουμε στο app engine. Όπως είπαμε και παραπάνω μας δίνεται URL για να δούμε κάποιο παλιότερο version της εφαρμογής. Επίσης σε αυτό το επίπεδο μπορούμε να δημιουργήσουμε και τον χώρο για να φιλοξενηθεί μια καινούρια εφαρμογή, στον οποίο χώρο μπορούμε να κάνουμε deploy την εφαρμογή όπως είδαμε παραπάνω, πατώντας το create application.

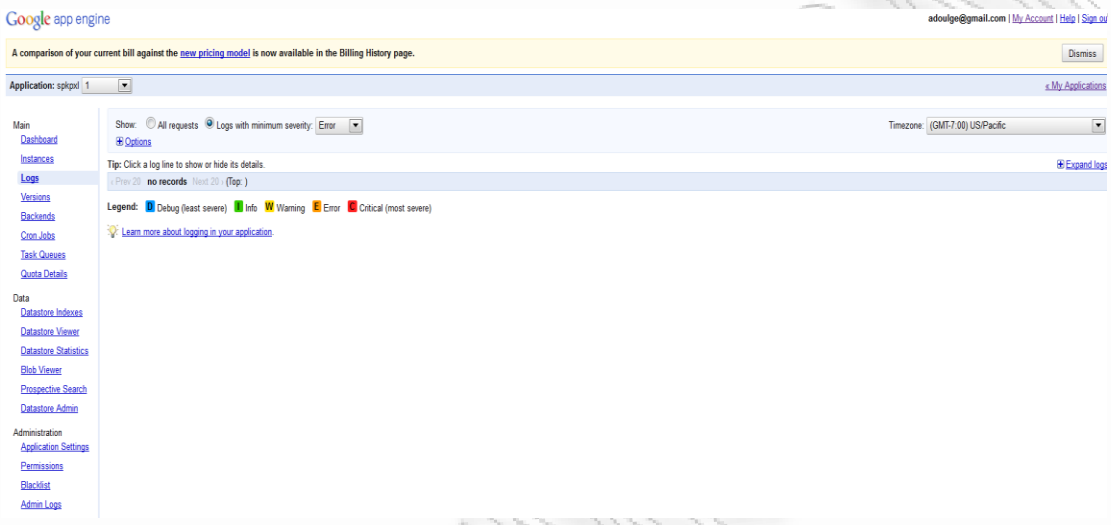
Στην συνέχεια μας εμφανίζεται το κεντρικό μενού του Google App engine με όλες τις δυνατότητες που έχουμε ήδη αναφέρει. Μας εμφανίζεται επιλογή να δούμε την επισκεψιμότητα της εφαρμογής μας με πολλά φίλτρα χρόνου, να δούμε πόση CPU στο server χρησιμοποιούμε, πόση μνήμη, τι χωρητικότητα έχουμε καταλάβει στο datastore με τις εγγραφές μας και πόσο μας απομένει ακόμα (ανάλογα με το επίπεδο συμβολαίου που έχουμε κάνει) κλπ. Γενικά στην αρχική σελίδα μας δίνεται η δυνατότητα να δούμε διάφορα σταστιστικά γενικά και συγκεντρωτικά.



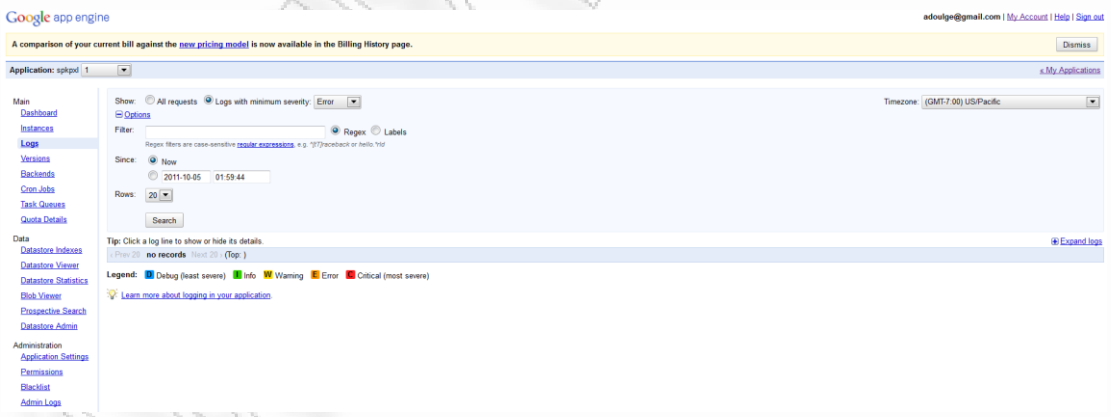
Σε περίπτωση που μεταβούμε στο Link της αρχικής σελίδας που λέει Instances βλέπουμε την παρακάτω σελίδα. Σε αυτή την σελίδα βλέπουμε πόσα Instances της εφαρμογής μας έχουν σηκωθεί μαζί με κάποια σταστικά για κάθε ένα απο αυτά όπως μνήμη που καταναλώνουν κλπ.



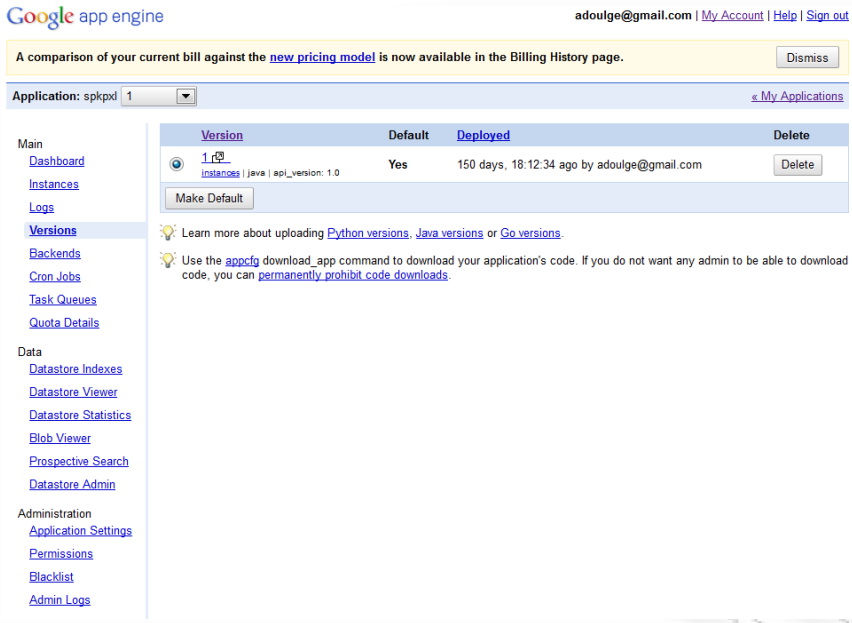
Στην παρακάτω σελίδα μπορούμε να πάμε πατώντας στο Link Logs και όπως υποδηλώνει και το όνομα του μας δείχνει πληροφορίες για Logs που έχει μαζέψει ο server για εμάς. Μπορούμε να δούμε για οποιο version της εφαρμογής θέλουμε Logs τα οποία είναι μαρκαρισμένα με κάποιο χαρακτηρισμό (debug, info, warning, error, critical).



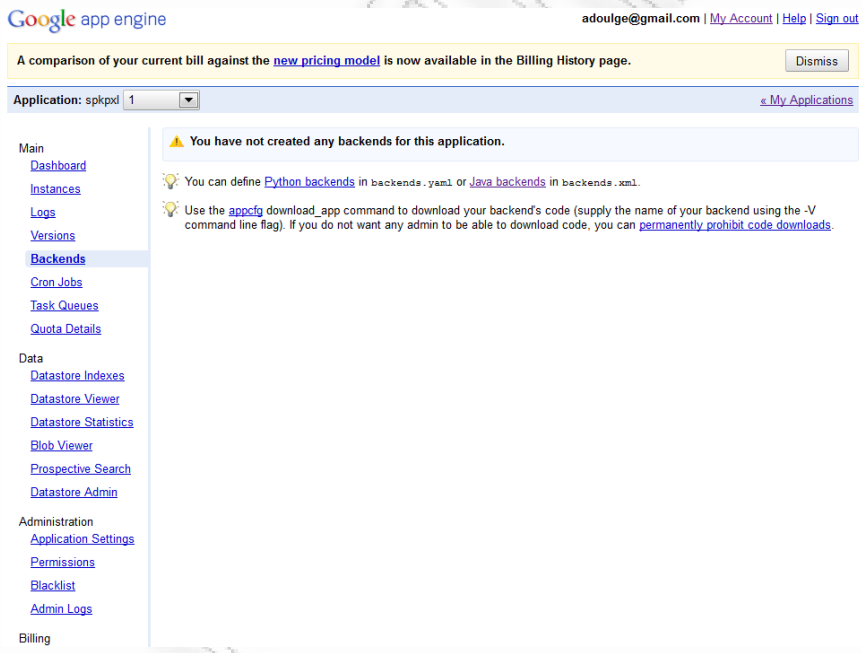
Επίσης μπορούμε να ορίσουμε εμείς χρονικά απο πότε μέχρι πότε θέλουμε να δούμε logs καθώς επίσης και άλλες παραμέτρους όπως φαίνεται στην παρακάτω σελίδα.



Στην συνέχεια επιλέγοντας απο το menu το Link versions μπορούμε να δούμε όλες τις version της εφαρμογής που έχουμε ανεβάσει στο Google App Engine και να σετάρουμε όποια απο αυτές θέλουμε σαν default η οποία θα είναι η Live έκδοση που θα βγαίνει έξω.

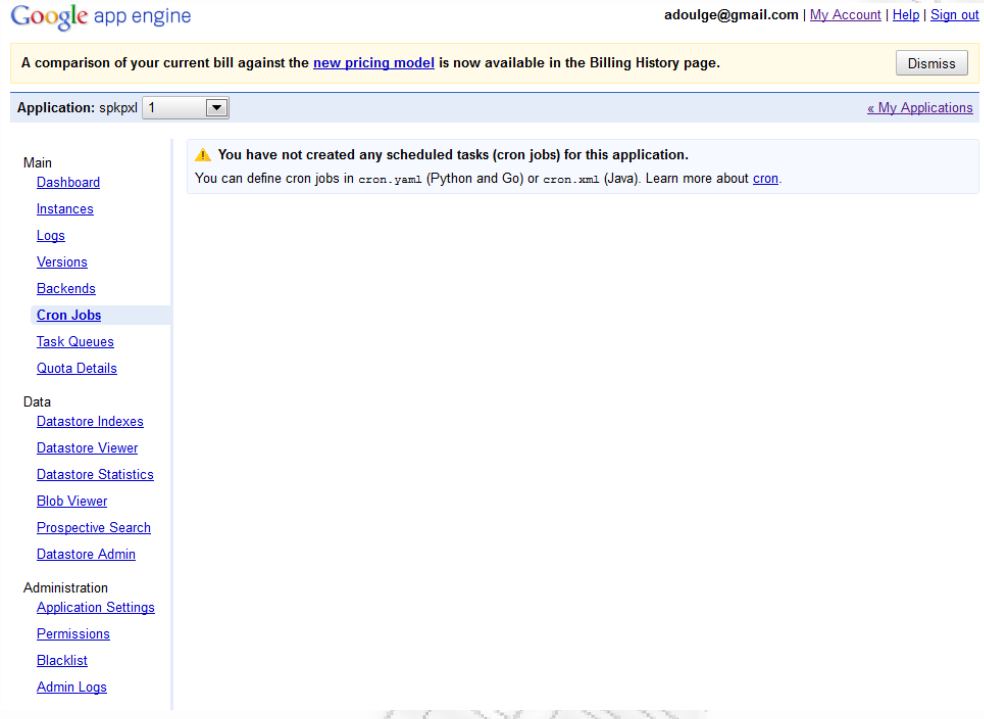


Άλλο ένα σημαντικό κομμάτι του GAE είναι τα backends τα οποία μπορούν να παραμετροποιηθούν στο Link backends του κεντρικού μενού, όπως φαίνεται στην παρακάτω σελίδα. Τα backends του app engine είναι Instances της εφαρμογής μου στα οποία δεν υπόκεινται οι περιορισμοί του GAE για memory (Μέχρι 1 GB) CPU (μέχρι 4,8 GHz) και χρονικό όριο εξυπηρέτησης request. Χρησιμοποιούνται απο εφαρμογές που χρειάζονται υψηλό performance και μεγάλη ταχύτητα στην εξυπηρέτηση requests.



Στην συνέχεια μπορούμε να δούμε την υπηρεσία που μας παρέχει το GAE για σημιουργία cron jobs όπως έχουμε ήδη αναφέρει. Όπως φαίνεται και παρακάτω

μπορούμε να ορίσουμε cron Jobs απλά πειράζοντας το cron.xml αρχείο που βρίσκεται στο WEB-INF φάκελο του project, αν γράφουμε σε java ή το cron.yaml για Python.



Άλλο ένα πολύ σημαντικό χαρακτηριστικό που παρέχει το Google App engine είναι η λεπτομερή καταγραφή στατιστικών του συστήματος όπως φαίνεται και παρακάτω.

Google app engine

adoulge@gmail.com | My Account | Help | Sign out

A comparison of your current bill against the new pricing model is now available in the Billing History page. Dismiss

Application: spkpxl 1 « My Applications

Main
- Dashboard
- Instances
- Logs
- Versions
- Backends
- Cron Jobs
- Task Queues
- **Quota Details**

Data
- Datastore Indexes
- Datastore Viewer
- Datastore Statistics
- Blob Viewer
- Prospective Search
- Datastore Admin

Administration
- Application Settings
- Permissions
- Blacklist
- Admin Logs

Billing
- Billing Settings
- Billing History

Resources
- Documentation
- FAQ
- Developer Forum
- Downloads

The quota details for this application are grouped by API and are listed below. If your application exceeds 50% of any particular quota halfway through the day, it may exceed the quota before the day is over. To learn more about how quotas work, read Understanding Quotas and Why is My App Over Quota?

Requests

Quotas are reset every 24 hours. Next reset: 20 hours

Resource	Daily Quota			Rate
CPU Time		0%	0.00 of 6.50 CPU hours	Okay
Requests		0%	0 of Unlimited	Okay
Outgoing Bandwidth		0%	0.00 of 1.00 GBytes	Okay
Incoming Bandwidth		0%	0.00 of 1.00 GBytes	Okay
Secure Requests		0%	0 of Unlimited	Okay
Secure Outgoing Bandwidth		0%	0.00 of 1.00 GBytes	Okay
Secure Incoming Bandwidth		0%	0.00 of 1.00 GBytes	Okay
Backend Usage		0%	$0.00 of $0.72	Okay

Storage

Datastore API Calls		0%	0 of Unlimited	Okay
Datastore Queries		0%	0 of Unlimited	Okay
Blobstore API Calls		0%	0 of Unlimited	Okay
Total Stored Data		0%	0.00 of 1.00 GBytes	Okay
Blobstore Stored Data		0%	0.00 of 1.00 GBytes	Okay
Data Sent to Datastore API		0%	0.00 of Unlimited GBytes	Okay
Data Received from Datastore API		0%	0.00 of Unlimited GBytes	Okay
Datastore CPU Time		0%	0.00 of Unlimited CPU hours	Okay
Datastore Entity Fetch Ops		0%	0 of Unlimited	Okay
Datastore Entity Put Ops		0%	0 of Unlimited	Okay
Datastore Entity Delete Ops		0%	0 of Unlimited	Okay
Datastore Index Write Ops		0%	0 of Unlimited	Okay
Datastore Query Ops		0%	0 of Unlimited	Okay
Datastore Key Fetch Ops		0%	0 of Unlimited	Okay
Datastore Id Allocation Ops		0%	0 of Unlimited	Okay
Number of Indexes		0%	0 of 200	Okay
Prospective Search Subscriptions		0%	0 of 10,000	Okay

Mail

Mail API Calls	0%	0 of 7,000	Okay
Recipients Emailed	0%	0 of 2,000	Okay
Admins Emailed	0%	0 of 5,000	Okay
Message Body Data Sent	0%	0.00 of 0.06 GBytes	Okay
Attachments Sent	0%	0 of 2,000	Okay
Attachment Data Sent	0%	0.00 of 0.10 GBytes	Okay

UrlFetch

UrlFetch API Calls	0%	0 of 657,084	Okay
UrlFetch Data Sent	0%	0.00 of 4.00 GBytes	Okay
UrlFetch Data Received	0%	0.00 of 4.00 GBytes	Okay

Image Manipulation

Image Manipulation API Calls	0%	0 of Unlimited	Okay
Data Sent to API	0%	0.00 of Unlimited GBytes	Okay
Data Received from API	0%	0.00 of Unlimited GBytes	Okay
Transformations executed	0%	0 of Unlimited	Okay

Memcache

Memcache API Calls	0%	0 of Unlimited	Okay
Data Sent to API	0%	0.00 of Unlimited GBytes	Okay
Data Received from API	0%	0.00 of Unlimited GBytes	Okay

XMPP

XMPP API Calls	0%	0 of 46,310,400	Okay
XMPP Data Sent	0%	0.00 of 1,046.00 GBytes	Okay
Recipients Messaged	0%	0 of 46,310,400	Okay
Invitations Sent	0%	0 of 100,000	Okay
Stanzas Sent	0%	0 of Unlimited	Okay

Channel

Channel API Calls	0%	0 of 46,310,400	Okay
Channels Created	0%	0 of 8,640	Okay
Channel Data Sent	0%	0.00 of 1,046.00 GBytes	Okay

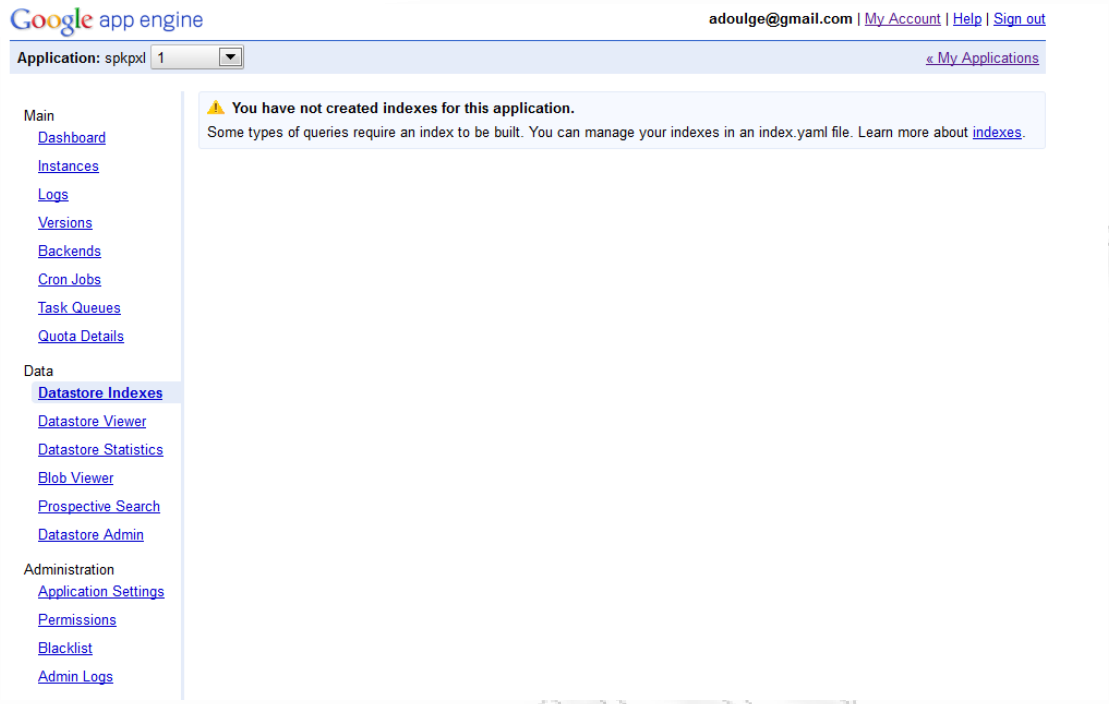
Task Queue

Task Queue API Calls	0%	0 of 100,000	Okay
Task Queue Stored Task Count	0%	0 of 1,000,000	Okay
Task Queue Stored Task Bytes	0%	0 of 104,857,600	Okay

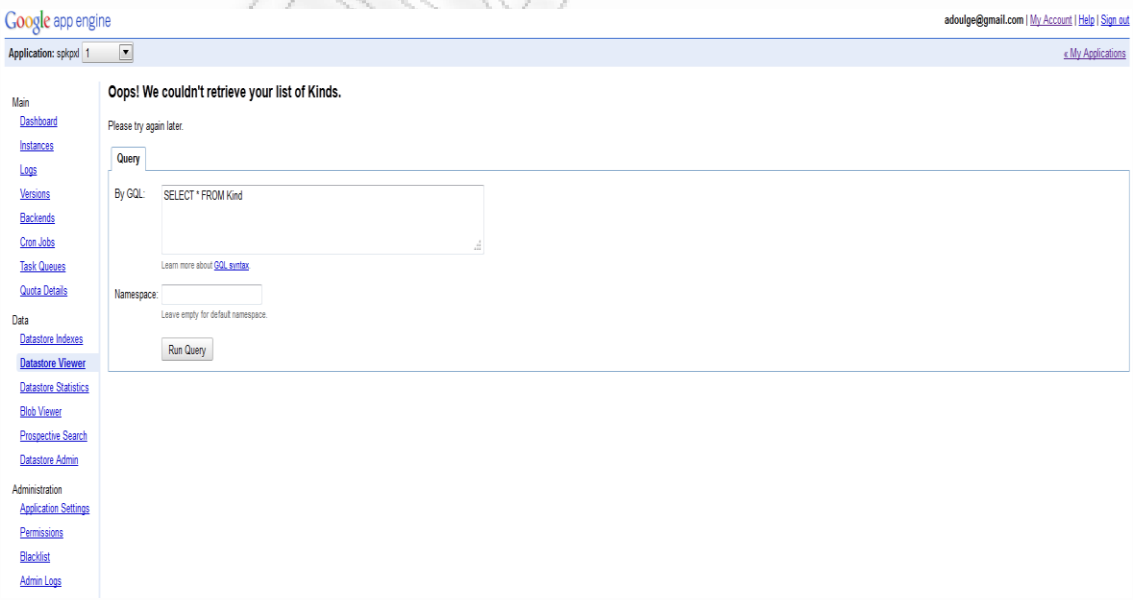
Deployments

Deployments	0%	0 of 1,000	Okay

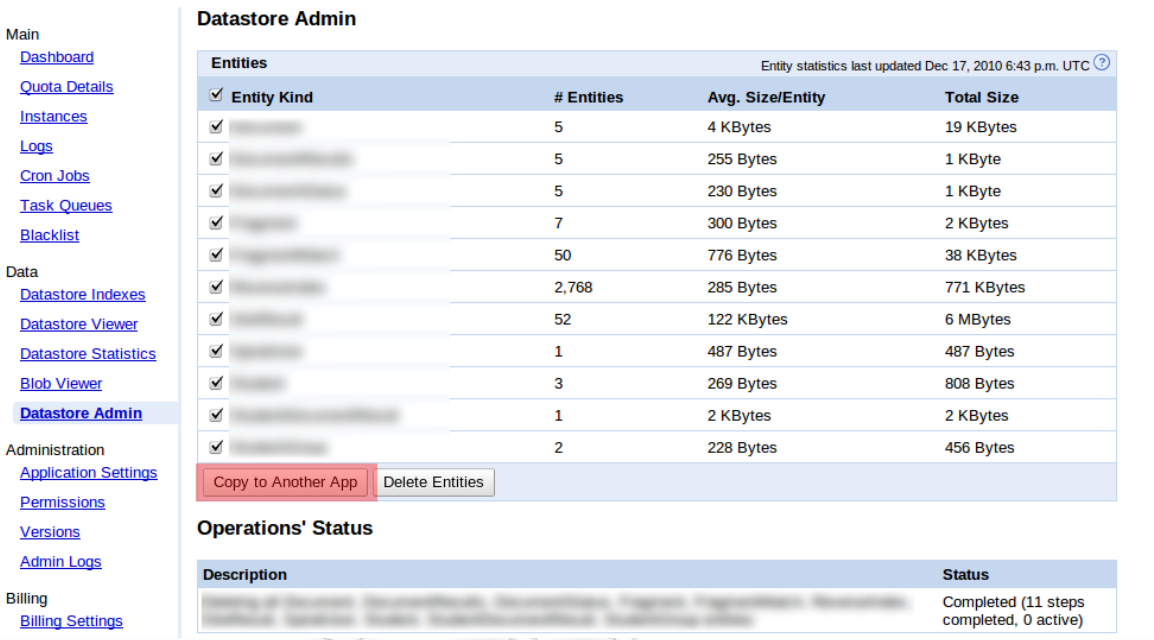
Μια πολύ σημαντική παροχή του Google App Engine είναι το datastore που έχουμε αναφέρει και παραπάνω. Όπως βλέπουμε και παρακάτω αν μεταβούμε στο section data δεξιά στο μενού του GAE θα δούμε τις διαθέσιμες επιλογές για να μπορέσουμε να διαχειριστούμε την βάση δεδομένων μας. Αρχικά βλέπουμε τα indexes που έχει δημιουργήσει το datastore.

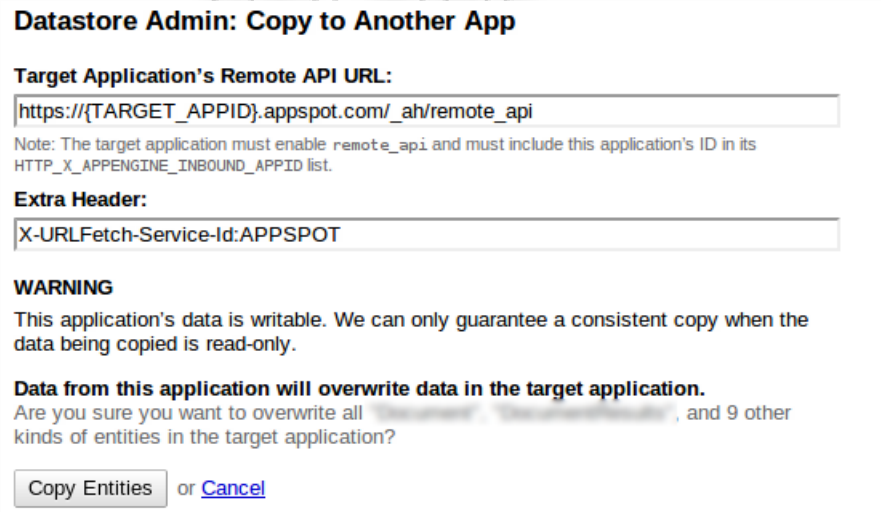


Στο επόμενο Link στην σειρά έχουμε το datastore viewer που όπως φαίνεται και παρακάτω είναι ένας editor για να μπορούμε να εκτελούμε queries στα entities που εχουμε στην βάση. Το Google παρέχει μια δικιά του query γλώσσα η οποία μοιάζει πολύ με την SQL και την ονομάζει GQL. Η μόνη διαφορά είναι ότι αντί να γίνονται τα queries πάνω σε πίνακες γίνονται πάνω σε entities μιας και το datastore χρησιμοποιεί την έννοια των entities αντί για των πινάκων.



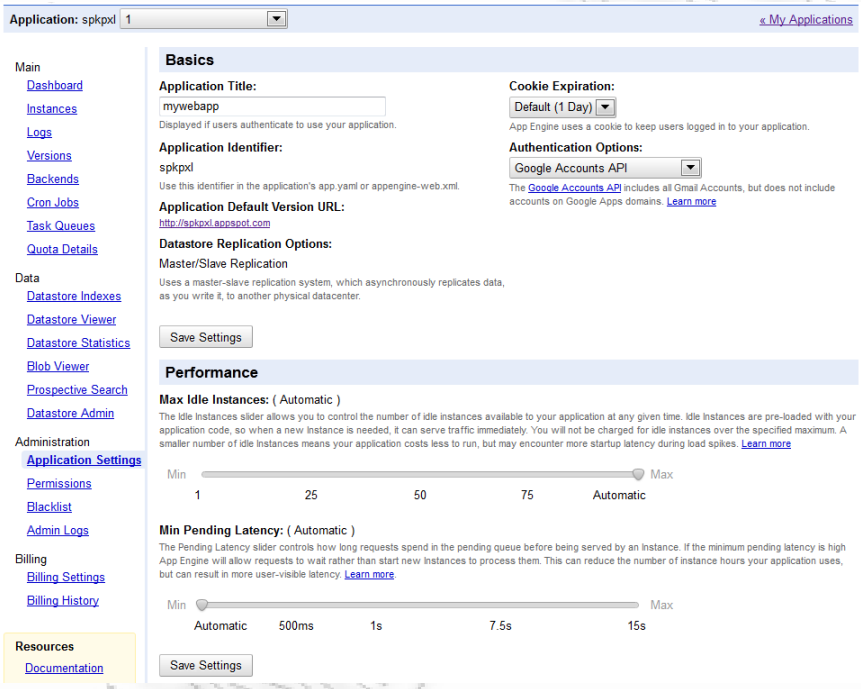
Το επόμενο Link (datastore statistics) μας δείχνει διάφορα σταστικά για το μέγεθος της βάσης δεδομένων που χρησιμοποιούμε, τον αριθμό των entities που έχουμε, το σύνολο των εγγραφών που έχουν αποθηκευτεί στο datastore κ.α. Τέλος υπάρχει το datastore admin στο οποίο ο διαχειριστής της εφαρμογής μπορεί να διαχειριστεί τα entities που έχει δημιουργήσει για την εφαρμογή. Επίσης δίνεται η δυνατότητα να τα αντιγράψει σε μια άλλη εφαρμογή οιυ βρίσκεται στο Google App Engine αρκεί να γνωρίζει το URL της άλλης εφαρμογής.



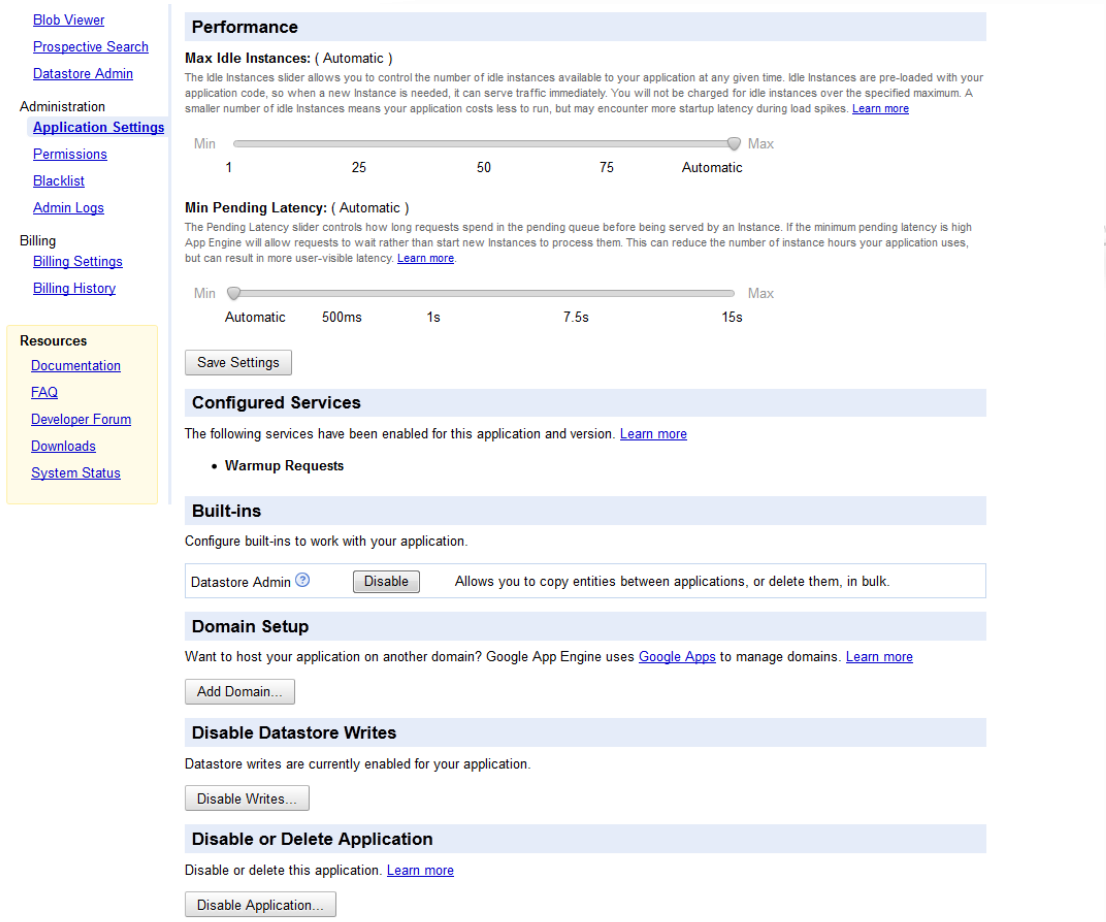


Στο επόμενο section sto menu του Google App Engine θα δούμε για το πως μπορούμε να διαχειριστούμε την εφαρμογή μας. Όπως φαίνεται παρακάτω όταν βρισκόμαστε στο link application settings μπορούμε να δούμε μερικά βασικά στοιχεία της εφαρμογής όπως το τίτλο που θα φαίνεται στον browser της εφαρμογής μας, το URL της, ποιο API χρησιμοποιούμε για authentication των χρηστών της εφαρμογής μας, ποιό θα είναι το lifetime των cookies που παράγονται απο την εφαρμογή και τέλος ποιο μοντέλο datastore χρησιμοποιούμε.

Στο επόμενο section έχουμε κάποιες πληροφορίες για το performance της εφαρμογής μας στο οποιό μπορούμε να ορίσουμε μέχρι πόσα idle instances Της εφαρμογής μπορούμε να έχουμε και μέχρι πόσο το πολύ μπορεί να αργήσει η εξυπηρέτηση ενός request. By default οι τιμές αυτών των slider είναι automatic πράγμα που σημαίνει ότι αποφασίζει το ίδιο το Google App Engine για το performance της εφαρμογής σύμφωνα με κάποια κριτήρια όπως επισκεψιμότητα, είδος συμβολαίου κλπ.

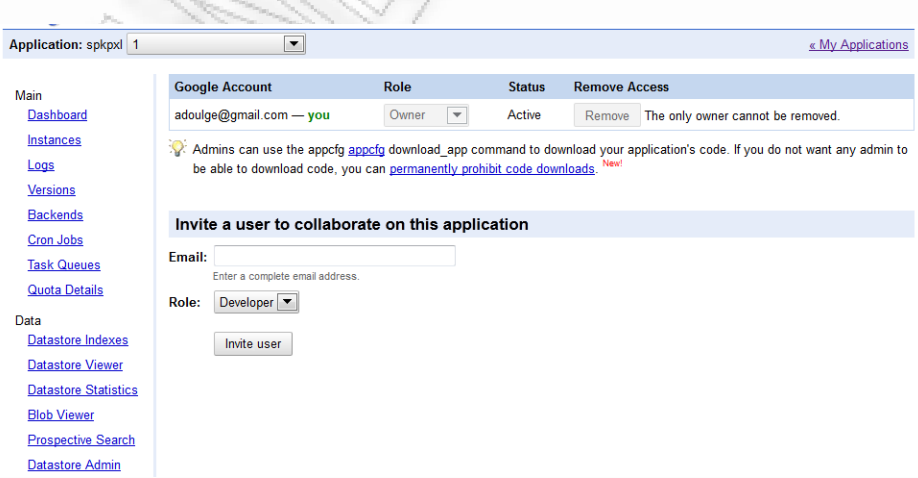


Στην συνέχεια βλέπουμε κάποια στοιχεία για τυχόν services που χρησιμοποιούμε. Πέρα απο HTTP requests τα οποία μπορεί να εξυπηρετήσει ο server μπορούμε να ορίσουμε και άλλα πρωτόκολλα όπως είναι email πρωτόκολλα και XMPP. Έτσι για παράδειγμα μπορούμε να έχουμε και mail server στο μηχανημα με email της παρακάτω μορφής *string@appid.appspotmail.com* το οποίο μπορεί να δέχεται και να στέλνει emails.

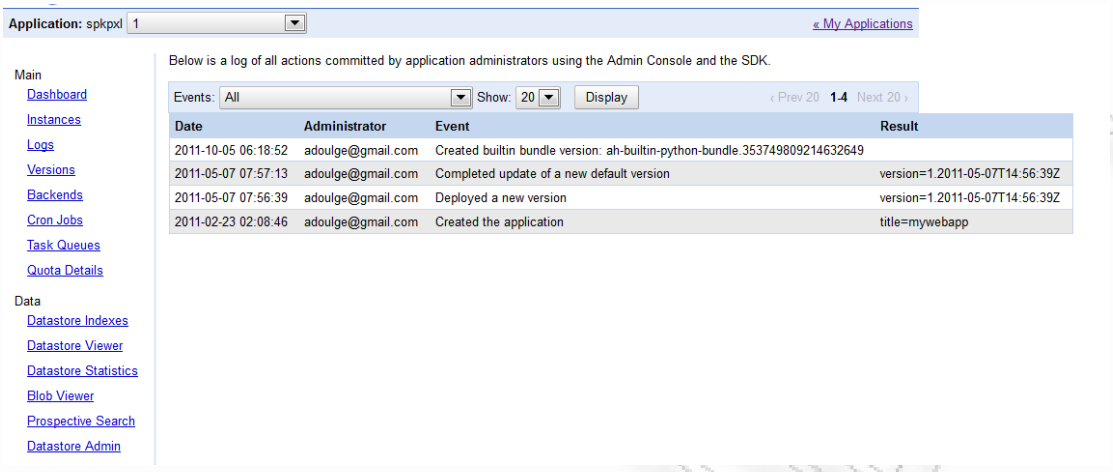


Τέλος όπως βλέπουμε μπορούμε να κάνουμε disable την δυνατότητα για να μπορούμε να αντιγράφουμε entities σε κάποια εξωτερική εφαρμογή η να τις διαγράφουμε πολλά entities μαζί. Επίσης μπορούμε να μεταφέρουμε όλη μας της εφαρμογή σε κάποιο εξωτερικό domain η ακόμη και να διαγράψουμε την εφαρμογή μας εντελώς.

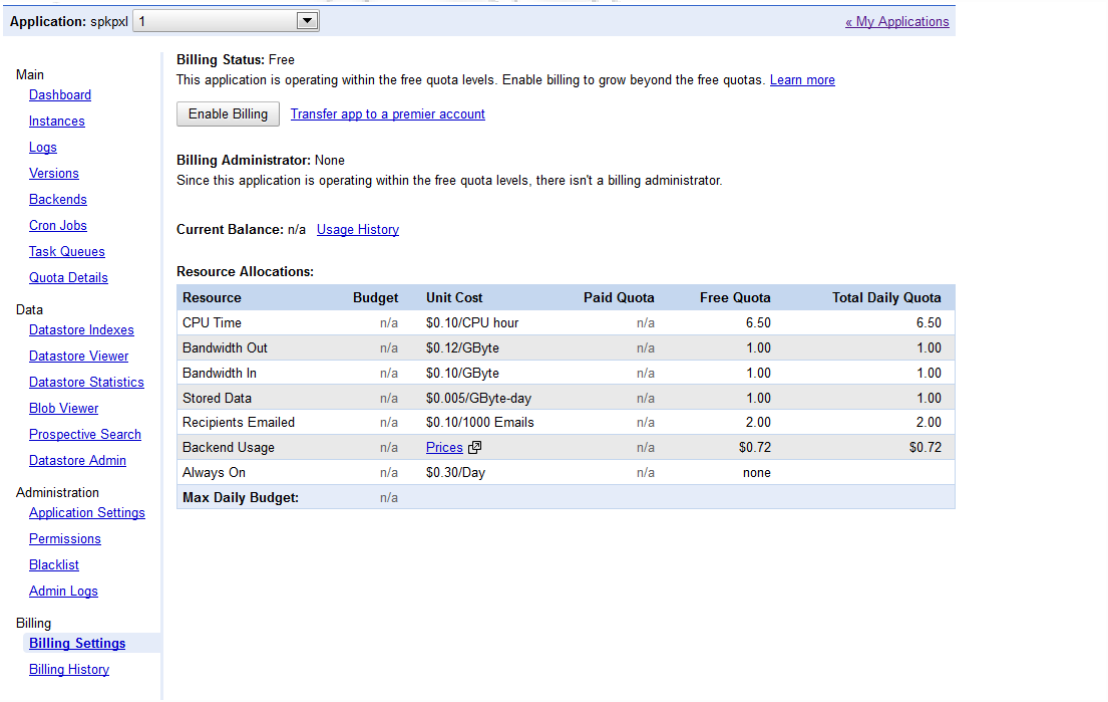
Το επόμενο Link που έχουμε αναφέρεται στα permissions και μπορούμε να δούμε κάθε συμβαλόμενος τι ρόλο έχει στο σύστημα και να το αλλάξουμε η να δημιουργήσουμε ένα καινούριο χρήστη και να του δώσουμε έναν συγκεκριμένο ρόλο με συγκεκριμένα privileges.



Τελευταίο κομμάτι του administration section που θα δείξουμε έιναι το Admin Logs τα στο οποίο όπως φαίνεται παρακάτω φαίνονται Logs για την εφαρμογή μας.



Το τελευταίο κομμάτι του Google App Engine που θα δούμε έχει να κάνει με το billing σύστημα των υπηρεσιών που παρέχει το GAE. Σε αυτό το σημείο οπως φαίνεται και παρακάτω μπορούμε να δούμε τον τυπο του συμβολαίου που έχουμε με την Google για την χρήση της υπηρεσίας που μας παρέχει μέσω του GAE. Όπως βλέπουμε μπορούμε να δούμε το τιμολόγιο και να φτιάξουμε στα μέτρα μας το πακέτο των υπηρεσίων που επιθυμούμε.

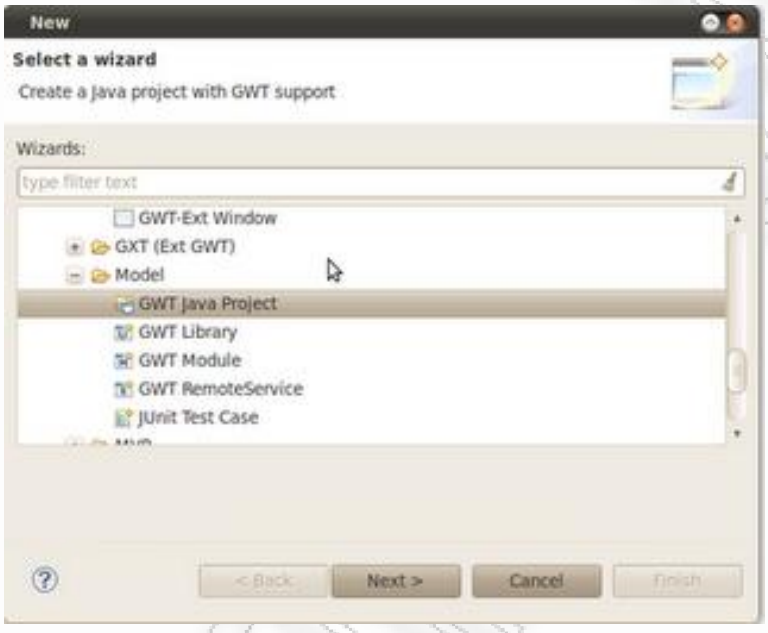


2.14 Datastore στην πράξη

Παρακάτω θα παροσυσιάσουμε ένα ολοκληρωμένο παράδειγμα χρήσης του Datastore API. Όπως ήδη έχουμε πει το datastore API χρησιμοποιείται για να κάνει δοσοληψίες με την βάση δεδομένων (αποθήκευση, διαγραφή και ανάκτηση δεδομένων). Το παράδειγμα που θα παρουσιάσουμε θα αποθηκεύει μια εγγραφή στο Datastore, θα ανακτά εγγραφές σύμφωνα με κάποιο query.

Σε αυτό το σημείο πρέπει να πούμε οτι χρησιμοποιούμε Eclipse Helios σαν περιβάλλον ανάπτυξης και GWT-2.2. Επομένως ξεκινάμε φτιάχνοντας ένα gwt project στο Eclipse.

New->Project->WindowBuilder->Model->GWT Java Project



Αφού τελειώσουμε και φτιάχτεί ο σκελετός του project μας είμαστε έτοιμοι να ξεκινήσουμε να γράφουμε τις κλάσεις μας. Πρώτα θα ξεκινήσουμε με απλή utility κλάση η οποία το μόνο που θα κάνει είναι να μας δίνει ένα instance του Persistence Manager, δηλαδή του manager που είναι υπεύθυνος για δοσοληψίες με το datastore.

PMF.java

```
package com.alex.datastore.db;
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;

public final class PMF {

private static final PersistenceManagerFactory
```

```
pmfInstance =
JDOHelper.getPersistenceManagerFactory("transactions-
optional");

private PMF() {}

public static PersistenceManagerFactory get() {
     return pmfInstance;
}
}
```

Όπως βλέπουμε δνε κάνει κάτι ιδιαίτερο αυτή κλάση παρα να μας επιστρέφει ένα Instance του Persistence Manager του datastore. Στην συνέχεια πάμε να φτιάξουμε το entity που θέλουμε να αποθηκεύσουμε μέσω του manager στο datastore.

HealthReport.java

```
package com.alex.datstore.db;
import java.util.Date;
import com.google.appengine.api.datastore.Key;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable(identityType                                =
IdentityType.APPLICATION)

public class HealthReport {

@PrimaryKey
@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
private Key key;

@Persistent
private String pinCode;

@Persistent
private String healthIncident;

@Persistent
private String status;

@Persistent
private Date reportDateTime;
```

```
public      HealthReport(String      pinCode,      String
healthIncident,String status, Date reportDateTime) {
     super();
     this.pinCode = pinCode;
     this.healthIncident = healthIncident;
     this.status = status;
     this.reportDateTime = reportDateTime;
}

public Key getKey() {
     return key;
}

public void setKey(Key key) {
     this.key = key;
}

public String getPinCode() {
     return pinCode;
}

public void setPinCode(String pinCode) {
     this.pinCode = pinCode;
}

public String getHealthIncident() {
     return healthIncident;
}

public void setHealthIncident(String healthIncident) {
     this.healthIncident = healthIncident;
}

public String getStatus() {
     return status;
}

public void setStatus(String status) {
     this.status = status;
}

public Date getReportDateTime() {
     return reportDateTime;
}

public void setReportDateTime(Date reportDateTime) {
     this.reportDateTime = reportDateTime;
}
}
```

Η παραπάνω κλάση περιέχει μερικά τα βασικά πεδία που θα έχει το entity που θέλουμε να αποθηκεύσουμε στο datastore και τις ανάλογες getter & setter μεθόδους για να δώσουμε και να παρουμε την εκάστοτε τιμή κάθε πεδίου. Στην συνέχεια θα δούμε πως μπορούμε να αποθηκεύσουμε ένα entity στο datastore. Για λόγους ευκολίας του παραδείγματος δεν θα χτίσουμε κάποιο UI επομένως θα φτιάξουμε απλά ένα servlet (PostHealthIncidentServlet.java) το οποίο στο URL όταν θα το χτυπάμε θα κάνει όλη την δουλειά. Τις παραμέτρους που θέλουμε να αποθηκεύσουμε θα τις περνάμε με τον ακόλουθο τρόπο στο URL του servlet

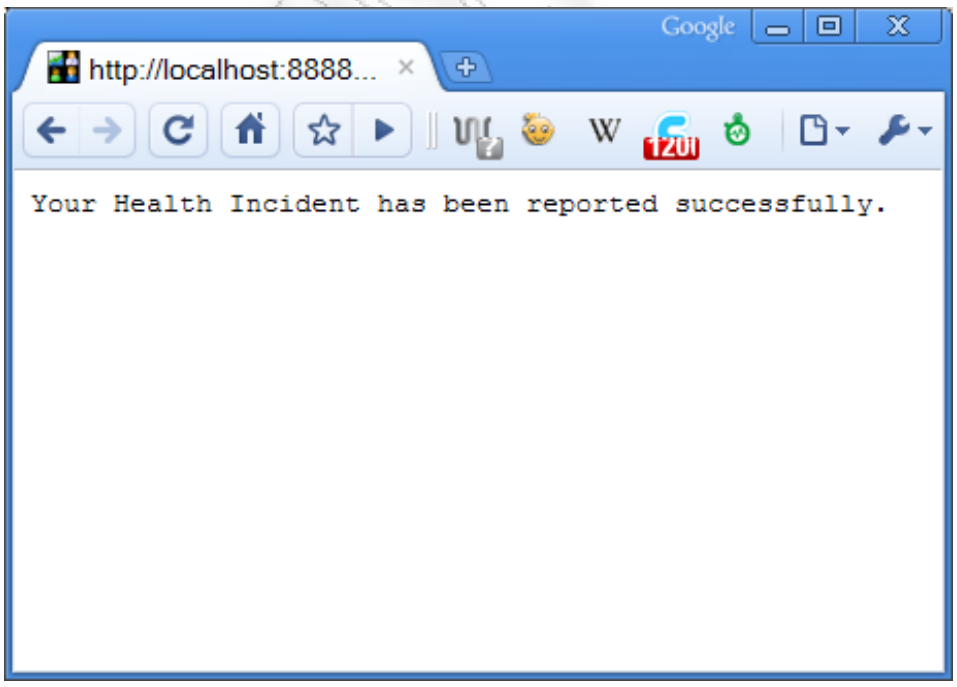
http://localhost:8888/posthealthincident?**healthincident**=Flu&**pincode**=400101

PostHealthIncidentServlet.java

```
package com.alex.datastore.db;
import java.io.IOException;
import java.util.Date;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.*;

@SuppressWarnings("serial")

public class PostHealthIncidentServlet extends
HttpServlet {

public static final Logger _logger =
Logger.getLogger(PostHealthIncidentServlet.class.getName(
));

@Override

protected void doGet(HttpServletRequest req,
HttpServletResponse resp)
throws ServletException, IOException {
doPost(req, resp);
}

public void doPost(HttpServletRequest req,
HttpServletResponse resp)
throws IOException { resp.setContentType("text/plain");
String strResponse = "";
String strHealthIncident = "";
String strPinCode = "";
try {
//DO ALL YOUR REQUIRED VALIDATIONS HERE AND THROW
EXCEPTION IF NEEDED
```

```
strHealthIncident                                                    =
(String)req.getParameter("healthincident");
strPinCode = (String)req.getParameter("pincode");
String strRecordStatus = "ACTIVE";
Date dt = new Date();
HealthReport HR = new HealthReport(strPinCode,
strHealthIncident,
strRecordStatus,
dt);
DBUtils.saveHealthReport(HR);
strResponse = "Your Health Incident has been reported
successfully.";
}
catch (Exception ex) {
_logger.severe("Error in saving Health Record : " +
strHealthIncident + "," + strPinCode + " : " +
ex.getMessage());
strResponse = "Error in saving Health Record via web.
Reason : " + ex.getMessage();
}
resp.getWriter().println(strResponse);
}
}
```

Σε αυτή την κλάση καλούμε τον Persistence Manager φτιάχνουμε το entity μας και μετά αποθηκεύουμε το entity στο datastore όπως είπαμε παραπάνω μέσω του URL (http://localhost:8888/posthealthincident?**healthincident**=Flu&**pincode**=400101) του servlet και τελος το servlet μας απαντάει αν η αποθήκευση ολοκληρώθηκε με επιτυχία.



Όπως βλέπουμε και παραπάνω στον κώδικα χρησιμοποιούμε ένα αντικείμενο της κλάσης DBUtils. Η κλάση DBUtils χρησιμοποιείται για να ανακτά και να αποθηκεύει δεδομένα απο και προς το datastore. Η περιγραφή είναι παρακάτω.

DBUtils.java

```
package com.alex.datastore.db;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.jdo.PersistenceManager;
import javax.jdo.Query;

public class DBUtils {

public static final Logger _logger =
Logger.getLogger(DBUtils.class.getName());

/*Currently we are hardcoding this list. But this could
also be retrieved from Database*/

public static String getHealthIncidentMasterList() throws
Exception {
     return "Flu,Cough,Cold";
}

/**
* This method persists a record to the database.
*/
public static void saveHealthReport(HealthReport
healthReport) throws Exception {

PersistenceManager pm =
PMF.get().getPersistenceManager();
try {
pm.makePersistent(healthReport); _logger.log(Level.INFO,
"Health Report has been saved");
} catch (Exception ex) {
_logger.log(Level.SEVERE,
"Could not save the Health Report. Reason : "
+ ex.getMessage());
throw ex;
} finally {
pm.close();
```

```
}
}

/**
* This method gets the count all health incidents in an
area (Pincode/Zipcode) for the current month
* @param healthIncident
* @param pinCode
* @return A Map containing the health incident name and
the number of cases reported for it in the current month
*/
public static Map<String, Integer>
getHealthIncidentCountForCurrentMonth(String
healthIncident, String pinCode) {

Map<String, Integer> _healthReport = new HashMap<String,
Integer>();
PersistenceManager pm = null;
//Get the current month and year
Calendar c = Calendar.getInstance();
int CurrentMonth = c.get(Calendar.MONTH);
int CurrentYear = c.get(Calendar.YEAR);
try {
//Determine if we need to generate data for only one
health Incident or ALL
String[] healthIncidents = {};
if (healthIncident.equalsIgnoreCase("ALL")) {
String strHealthIncidents =
getHealthIncidentMasterList();
healthIncidents = strHealthIncidents.split(",");
}
else {
healthIncidents = new String[]{healthIncident};
}
pm = PMF.get().getPersistenceManager();
Query query = null;
//If Pincode (Zipcode) is ALL, we need to retrieve all
the records irrespective of Pincode
if (pinCode.equalsIgnoreCase("ALL")) {
//Form the query
query = pm.newQuery(HealthReport.class, " healthIncident
== paramHealthIncident && reportDateTime >=
paramStartDate && reportDateTime < paramEndDate && status
== paramStatus");
// declare parameters used above paramStartDate,
java.util.Date paramEndDate, String paramStatus");
}
else {
query = pm.newQuery(HealthReport.class, " healthIncident
== paramHealthIncident && pinCode == paramPinCode &&
```

```
reportDateTime >= paramStartDate && reportDateTime
<paramEndDate && status == paramStatus");
// declare params used above
query.declareParameters("String paramHealthIncident,
String paramPinCode, java.util.Date paramStartDate,
java.util.Date paramEndDate, String paramStatus");
}
/*For each health incident (i.e. Cold Flu Cough),
retrieve the records*/
for (int i = 0; i < healthIncidents.length; i++) {
int healthIncidentCount = 0;
/*Set the From and To Dates i.e. 1st of the month and 1st
day of next month*/
Calendar _cal1 = Calendar.getInstance();
_cal1.set(CurrentYear, CurrentMonth, 1);
Calendar _cal2 = Calendar.getInstance();
_cal2.set(CurrentYear,CurrentMonth+1,1);
List<HealthReport> codes = null;
if (pinCode.equalsIgnoreCase("ALL")) {
/*Execute the query by passing in actual data for the
filters*/
codes = (List<HealthReport>)
query.executeWithArray(healthIncidents[i],_cal1.getTime()
,_cal2.getTime(),"ACTIVE");
}
else {
codes = (List<HealthReport>)
query.executeWithArray(healthIncidents[i], pinCode,
_cal1.getTime(),_cal2.getTime(),"ACTIVE");
}
//Iterate through the results and increment the count
for (Iterator iterator = codes.iterator();
iterator.hasNext();) {
HealthReport _report = (HealthReport) iterator.next();
healthIncidentCount++;
}
//Put the record in the Map data structure
_healthReport.put(healthIncidents[i], new
Integer(healthIncidentCount));
}
return _healthReport;
} catch (Exception ex) {
return null;
} finally {
pm.close();
}
```

Στην παραπάνω κλάση περιέχονται όλες οι μέθοδοι που έχουν να κάνουμε με δοσοληψίες με την βάση δεδομένων, δηλαδή αποθήκευση (saveHealthReport) και ανάκτηση (getHealthIncidentCountForCurrentMonth).

Στην συνέχει θα δούμε πως μπορούμε να δούμε όλα τα entities και τις εγγραφές που έχουμε κάνει στο datastore τοπικά για να σιγουρευτούμε ότι όλα λειτουργούν σωστά πριν ανεβάσουμε την εφαρμογή μας στο Google App Engine. Έτσι αν πληκτρολογήσουμε στον browser `http://localhost:8888/_ah/admin` θα μεταφερθούμε στο backend του datastore και θα δούμε αρχικά όπως φαίνεται παρακάτω μία λίστα με όλα τα entities που έχουμε.

Για κάθε entity που επιλέγουμε μπορούμε να δούμε τις εγγραφές που περιέχει και να προσθέσουμε η να διαγράψουμε όποια θέλουμε.

3. Γενικά για AJAX

Η τεχνολογία AJAX άλλαξε κατα πολύ τοον τρόπο που σχεδιάζοναται και υλοποιούνται εφαρμογές στο διαδίκτυο μιας και έσπασε τα δεσμά του browser-dependency. Ακόμη εισήγαγε για πρώτη φορά ένα πολύ χρήσιμο feature το οποίο έδινε την δυνατότητα στους browser να κάνουν update στο περιεχόμενο των σελίδων τους χωρίς αυτό να φαίνεται στον χρήστη, χωρίς δηλαδή να στέλνουν όλη τη σελίδα που έγινε update στον server και μετά ο να την στέλνει πίσω στον browser updated ο server.

Επειδή το GWT υποστηρίζει την τεχνολογία AJAX είναι πολύ εύκολο οι εφαρμογές που γράφονται με αυτό το framework να υποστηρίζονται απο όλους τους γνωστούς browser αλλά και απο ενδεχόμενους νέους. Αν τη συγκρίνουμε με το Silverlight και το Flash θα δούμε ότι αυτές οι τεχνολογίες όντως δίνουν την δυνατότητα δημιουργίας ευέλικτων web application σχετικά γρήγορα όμως η εξάρτιση τους απο το περιβάλλον που βρίσκεται εκτός του browser, του λειτουργικού συστήματος δηλαδή, είναι αρκετά μεγάλη και αυτό έχει σαν αποτέλεσμα αρκετές φορές την καθυστέρηση αυτών των εφαρμογών. Το GWT πέρα απο cross-browser είναι και cross-platform με την έννοια ότι παίζει ακριβώς το ίδιο σε Linux, Windows, MacOS κλπ.

3.1 Rich Interfaces με Widgets και Panels

Το GWT παρέχει την δυνατότητα στο προγραμματιστή να μπορεί να κατασευάζει εύκολα και γρήγορα πλούσια και ευέλικτα UI (user interfaces). Υπάρχει μια βιβλιοθήκη με widgets και panels την οποία μπορούμε να χρησιμοποιήσουμε στον κώδικα για την δημιουργία AJAX εφαρμογών. Αυτά τα εργαλεία που μας παρέχει αυτή η βιβλιοθήκη είναι κατασκευασμένα σε HTML με Javascript για να μπορούν να εξυπηρετήσουν διάφορα events. Όταν η εφαρμογή μας γίνεται compile απο Java σε Javascript, ο browser απλά μπορεί να τα εμφανίσει χωρίς κανένα plugin και χωρίς το JRE.

Τα widgets του GWT μας δίνουν προγραμματιστικό έλεγχο πάνω στα πολύ καλά ορισμένα στοιχεία του. Πέρα απο τα κλασσικά widgets τα οποία υλοποιούν HTML tags υπάρχουν και αρκετά πιο περίπλοκα widgets τα οποία δεν υπάρχουν σαν μεμονωμένα HTML tags όπως είναι το tree widget το οποίο συνήθως συναντάται σε desktop εφαρμογές.

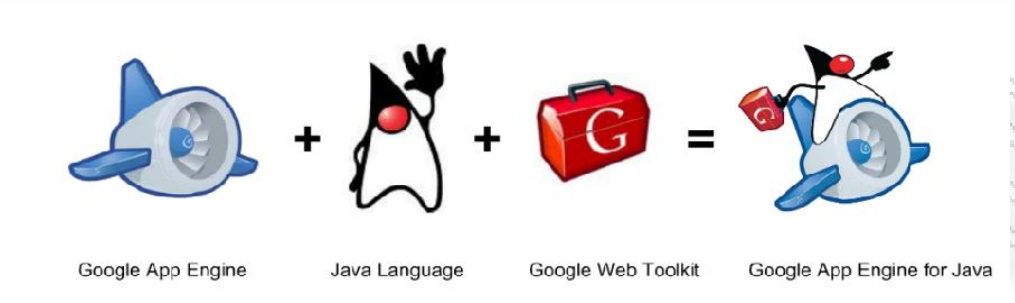
Το εργαλείο επίσης παρέχει panels τα οποία βοηθούν στην δημιουργία του layout της εφαρμογής. Τα panels είναι και αυτά ορισμένα με πολύ αυστηρούς κανόνες ώστε να δείχνουν τα widgets που περιέχουν μέσα τους με τον ίδιο τρόπο σε οποιδήποτε web browser.

3.2 Ασύγχρονη επικοινωνία μεσω AJAX

Οι AJAX εφαρμογές βελτιώνουν σε επίπεδο performance τις επιδόσεις του web server. Αυτό το επιτυγχάνουν με τον εξής τρόπο, αντί να φορτώνουν την σελίδα κάθε φορά που γίνεται μια αλλαγή φορτώνουν μόνο το συγκεκριμένο στοιχείο της σελίδα στο οποίο έγιναν οι αλλαγές.

Το GWT παρέχει μια βιβλιοθήκη για αποστολή και λήψη δεδομένων προς τον server ασύγχρονα, καθώς επίσης και μια υλοποίηση για RPCs για κλήση μεθόδων στον server σαν να ήταν σε κάποιο τοπικό μηχάνημα.

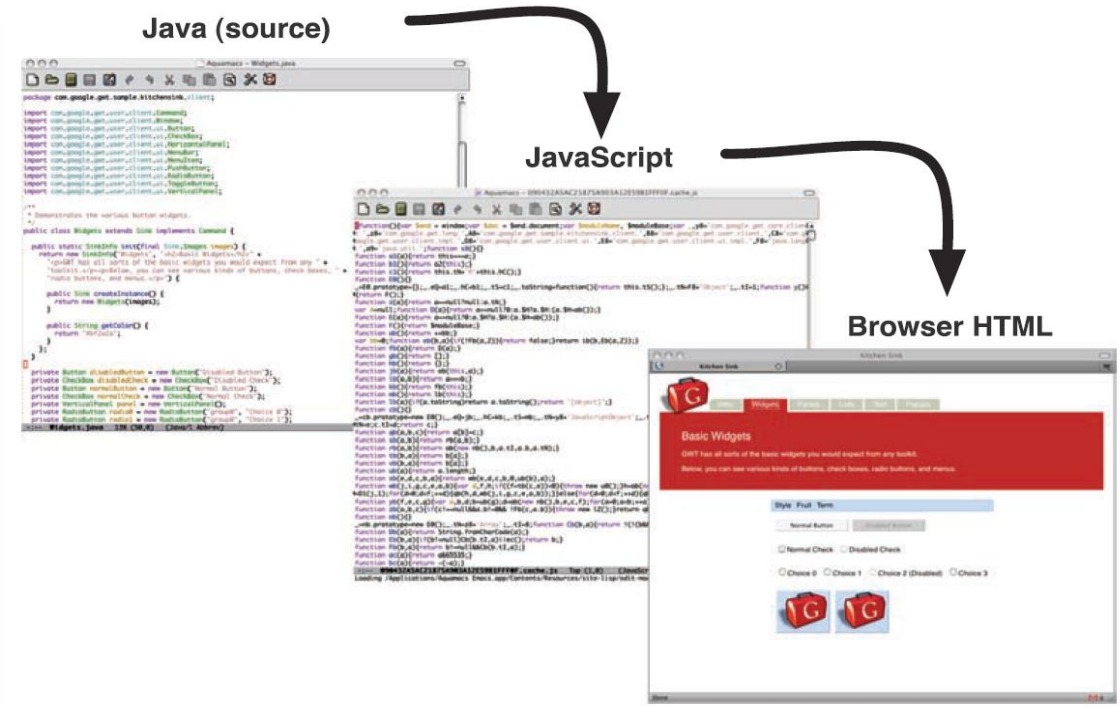
4. Γενικά για το GWT



Το GWT (Google Web Toolkit) είναι ένα εργαλέιο ανάπτυξης εφαρμογών (development toolkit) διαδικτύου. Έχει κατασκευαστεί απο την Google και βασικός σκοπός του είναι η δυνατότητα δημιουργίας διαδικτυακών εφαρμογών οι οποίες θα παρέχουν υψηλή διαθεσιμότητα και ταχύτητα. Το GWT χρησιμοποιεί AJAX requests (XMLHttpRequest) για να επικοινωνήσει ο client με τον server.

Ο προγραμματισμός διακικτυακών εφαρμογών με AJAX είναι αρκετά επίπονη γιατί δεν είναι browser independent και το debugging σε διαφορετικά περιβάλλοντα είναι προβληματικό. Έτσι αρχικά δημιουργήθηκαν μια σειρά απο βιβλιοθήκες όπως Dojo, Ext JS, Yahoo User Interface (YUI) κ.α για να βοηθήσουν να ξεπεραστούν αυτά τα προβλήματα. Όλες οι παραπάνω προσεγγίσεις είναι ορθές αλλά το GWT είναι κάτι διαφορετικό.

Το GWT είναι ένας Java to Javascript cross-compiler. Στην ουσία μεταγλωττίζει java κώδικα σε Javascript για να μπορέσει να τρέξει σε web browser (σχήμα 1). Αρχικά κυκλοφόρησε τον Μάιο του 2006 και στη συνέχεια έγινε open source το 2007. Κατασκευάστηκε με αρχική σκέψη να παρέχει ένα toolkit με το οποίο ο προγραμματιστής να μπορεί να φτιάχνει cross-browser web applications, οι οποίες θα μπορούν να γίνουν debug, να μπορούν να τεσταριστούν και να υποστηρίζουν Internationalization .



Το GWT κάνει την συγγραφή AJAX application Πολύ πιο έυκολη για τον προγραμματιστή και πολύ πιο ευέλικτη την τελική εφαρμογή για τον χρήστη. Παρέχει την δυνατότητα για δημιουργία Rich Internet Applications (RIAs) με την έννοια ότι ο χρήστης μπορεί τοπικά να κάνει υπολογισμούς χωρίς να χρειάζεται να επικοινωνήσεις με τον server για κάθε αλλαγή. Αυτό έχει σαν αποτέλεσμα ο τελικός χρήστης να έχει μια πολύ πιο responsive εφαρμογή με ένα έυρος απο έυχρηστα UI elements.

4.1 Τι περιλαμβάνει το GWT

Το GWT περιλαμβάνει ένα πλήθος εργαλείων για την εύκολη συγγραφή AJAX εφαρμογών. Παρακάτω γίνεται αναφορά και σύντομη περιγραφή τους.

- **GWT Compiler**

Ο compiler είναι το εργαλείο που είναι υπεύθυνο για την μεταγλώττιση του Java κώδικα σε Javascript. Αυτό επιτυγχάνεται ορίζοντας compile tasks στα Module απο τα οποία αποτελείται το project μας. Αντίθετα με τα άλλα εργαλεία συγγραφής AJAX εφαρμογών, το GWT παρέχει εύκολα integration με όλους τους γνωστούς web browser. Αυτό επιτυγχάνεται δηλώνοντας στον client το target των core classes με αποτέλεσμα ο κάθε web browser να κοιτάζει στον κώδικα που μπορεί να τρέξει και να μην χρειάζεται να τον κατεβάσει εκείνη τη στιγμή.

- **User Interface Layer**

To User interface layer αποτελεί την βάση του intelligent compilation system που παράγει cross-browser UI. Το GWT UI layer παρέχει μια μεγάλη ποικιλία panels, trees, grid, κ.α. τα οποία παρέχουν μεγάλες ευκολίες στον προγραμματιστή.

- **Remote Procedure calls**

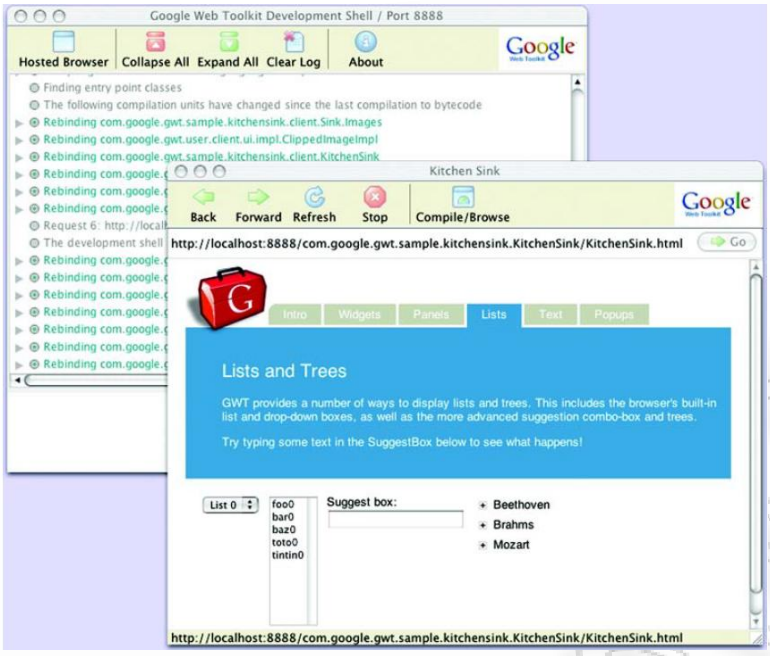
Με το GWT RPC μηχανισμό παρέχεται η δυνατότητα επικοινωνίας με τον server με serialization και deserialization των Java Objects στον server τα οποία μπορούν μετά να κληθούν ασύγχρονα απο τον client. Αυτό επιτυγχάνεται απο τον compiler αρχικά παράγοντας κώδικα κατά compilation step για να μπορέσει να διαχειριστεί το serialization σε χαμηλό επίπεδο. Τα serialized objects πέρνουν κάποιο version και γίνονται mapped κατα την διάρκεια του compilation. Αυτό έχει σαν αποτέλεσμα να υπάρχει μια συνοχή στο version των objects που ανταλλάσουν ο server με τον client.

- **Additional Utilities**

Πέρα απο τα core elements το GWT παρέχει και μια μεγάλη ποικιλία απο utilities τα οποία βοηθούν την ανάπτυξη εφαρμογών. Αυτά περιλαμβάνουν εργαλεία για εναλλακτικούς τρόπους επικοινωνίας με τον server, internationalization, διαχείρισης ιστορικού συστήματος (history management systems) και testing.

- **GWT Shell**

Το GWT Shell σου παρέχει την δυνατότητα να τεστάρεις την εφαρμογή καθώς τρέχεις native Java bytecode. Αυτό δίνει την δυνατότητα στον προγραμματιστή να χρησιμοποιεί profilers, να κανει step-through debugging, ενώ μέω του Hosted mode browser ο οποίος περιέχει και έναν Apache web server παρέχουν την δυνατότητα τεσταριστούν τα compiled Javascript αρχεία με JUnit.



(Πηγή: «Manning GWT in Practice», Robert T. Cooper, Charlie E. Collins, Manning publications).

Βασική δομή GWT Project

Ένα GWT Project αποτελείται απο τα εξής, modules, host pages και entry point classes.

- **Modules**

Το project ιεραρχείται σε modules. Κάθε module ορίζεται απο ένα XML αρχείο το οποίο δείχνει που βρίσκονται οι κλάσεις του εκάστοτε Module. Αυτό το αρχείο περιέχει ακόμα πληροφοριές για τα inherited modules, compiler plugins, entry points και servlet deployments. (Σχήμα 2)

```
<module>
   <inherits name='com.google.gwt.user.User'/>
   <entry-point
      class='com.manning.gwtip.calculator.client.Calculator'/>
   <stylesheet
      src='com.manning.gwtip.calculator.style.css'/>
</module>
```

1 Inherit core GWT classes
2 Define EntryPoint
3 Specify CSS file to use

Σχήμα 2 Παράδειγμα module definition.(Πηγή: «Manning GWT in Practice», Robert T. Cooper, Charlie E. Collins, Manning publications).

Host Pages

Για να τρέξει μια GWT εφαρμογή πρέπει να ξεκινήσει απο μια HTML σελίδα. Οι host pages περιέχουν κάποια elements για να μπορέσει να ενεργοποιηθεί και να τρέξει η εφαρμογή, να δούμε το αποτέλεσμα (html page) στον web browser. Αρχικά σε αυτές τις σελίδες υπάρχει μια αναφορά σε ένα nocache Javascript αρχείο το οποίο παράγεται απο το GWT για κάθε module. Αυτό το αρχείο ελέγχει την έκδοση του Module στον χρήστη και φορτώνει την αντίστοιχη εφαρμογή/κομμάτι εφαρμογής. (Σχήμα 3)

```
<html>
    <head>
        <title>Wrapper HTML for Calculator</title>
    </head>
    <body>
        <script language="javascript"
            src=
"com.manning.gwtip.calculator.Calculator.nocache.js"
            >
            </script>
        <iframe id="__gwt_historyFrame"
                style="width:0;height:0;border:0"></iframe>
        <h1>Calculator</h1>

    </body>
</html>
```

1 Bootstrap JavaScript file

Special invisible iframe for history management

(Σχήμα 3) Παράδειγμα HTML host page (Πηγή: «Manning GWT in Practice», Robert T. Cooper, Charlie E. Collins, Manning publications).

Entry Point Classes

Το entry point class είναι ένα παράδειγμα client-side κλάσης η οποία κάνει implement το com.google.gwt.core.client.EntryPoint interface μέσα στο οποίο ορίζεται η μέθοδος onModuleLoad(). Αυτή η μέθοδος είναι το αντίστοιχο της main() που υπάρχει στην Java. Η μέθοδος αυτή είναι απλή και βασικό στοιχείο είναι η χρήση της GWT κλάσης RootPanel. Μέσω αυτή της κλάσης έχουμε πρόσβαση με την HTML host page, απλά καλώντας την μέθοδο get αυτής της κλάσης (RootPanel.get()) η οποία μας επιστρέφει τον default container. Αν θέλουμε να μας επιστραφεί ο container ενός συγκεκριμένου widget απλά καλούμε RootPanel.get("Element ID"), όπου Element ID είναι το όνομα του element στην HTML σελίδα. Σχήμα 4.

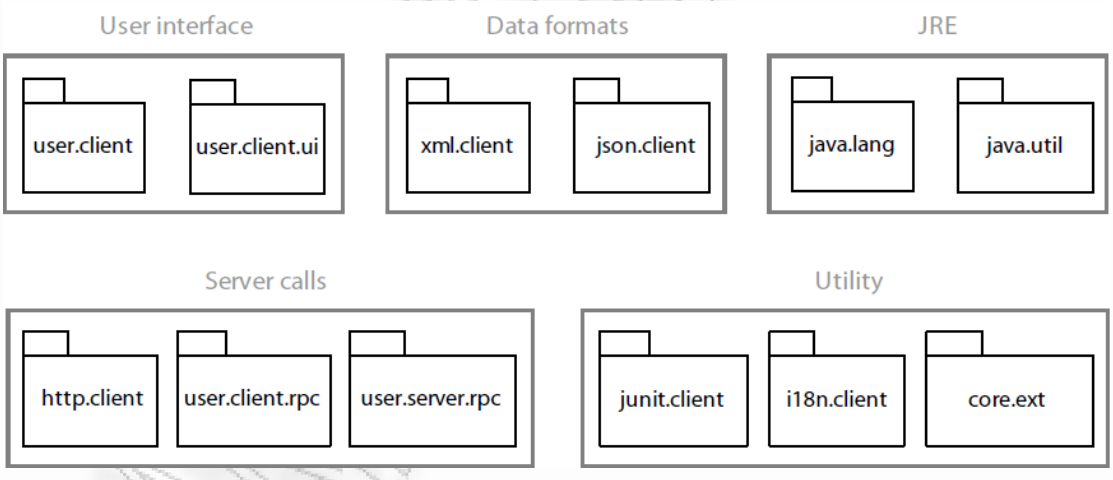
```
public class Calculator implements EntryPoint {
    public void onModuleLoad() {
        RootPanel.get().add(
            new CalculatorWidget("calculator"));
    }
}
```

1 Add CalculatorWidget to <body>

(Σχήμα 4) Παράδειγμα entry point class. (Πηγή: «Manning GWT in Practice», Robert T. Cooper, Charlie E. Collins, Manning publications).

4.2 Πακέτα GWT

Αν και βασικό πλεονέκτημα GWT προέρχεται από τη χρήση της Java και τα εργαλεία για να χτίσεις web applications, πρέπει να επισημάνουμε την ποικιλία των πακέτων που μας παρέχει το GWT για την διευκόλυνση στη δημιουργία RIAs (Rich Internet Applications). Τα πακέτα αυτά βρίσκονται μέσα στο gwt-user.jar το οποίο βρίσκεται στο Installation directory του GWT. Όλες οι GWT εφαρμογές πρέπει να τρέξουν αυτο το αρχείο όταν βρίσκονται σε hosted mode γιατί περιέχει ζωτικής σημασίας πληροφοριά για να μπορέσει να τρέξει η εφαρμογή. Κατα το compilation step χρησιμοποιείται αυτό το jar για να παραχθεί Javascript απο την Java.



(Πηγή: «Manning GWT in Practice», Robert T. Cooper, Charlie E. Collins, Manning publications).

Τα πακέτα για το User interface, τα οποία αποτελούν και το μεγαλύτερο κομμάτι πακέτων του GWT, έχουν να κάνουν με την δυναμική δημιουργία UI συστατικών. Όπως φαίνεται και στο παραπάνω σχήμα υπάρχουν δύο πακέτα τα οποία περιέχουν κλάσεις για την δημιουργία UI.

- **com.google.gwt.user.client**

  Αυτό το πακέτο έχει βασική πληροφορία για επικοινωνία με APIs των γνωστών web browser για την δημιουργία δυναμικών και cross-browser web applications.

- **com.google.gwt.user.client.ui**

  Αυτό το πακέτο περιέχει το framework για την δημιουργία UI και είναι αρκετά συναφή με το Java's Abstract Window Toolkit (AWT).

Άλλο ένα πακέτο είναι αυτό που έχει να κάνει με την επικοινωνία με τον server, τα λεγόμενα server calls τα οποία μας παρέχουν την δυνατότητα επικοινωνίας client-server. Αυτά τα πακέτα περιλμβάνουν όλη την λειτουργικότητα για την δημιουργία δυναμικών UI , ασύγχρονης επικοινωνίας client-server που χρειάζεται μι βιβλιοθήκη για την δημιουργία AJAX εφαρμογών.

- **com.google.gwt.http.client**

  Αυτό είναι ένα HTTP πακέτο το οποίο δημιουργεί έναν browser-independent wrapper πάνω στο XMLHttpRequest το οποίο είναι το αντικείμενο που χρησιμοποιούν για ασύγχρονη επικοινωνία.

- **com.google.gwt.user.client.rpc & com.google.gwt.user.server.rpc**

  Αυτά τα πακέτα δίνουν την δυνατότητα στον client να επικοινωνεί με τον server με την χρήση RPCs (Remote Procedure Call). Αυτές οι βιβλιοθήκες βασίζονται σε RPC relay calls σε Java interfaces στον server χωρίς να ενδιαφέρονται για τις λεπτομέρειες του πρωτόκολλου.

Άλλη μια κατηγορία πακέτων είναι αυτά που δίνουν την δυνατότητα parsing και constructing data formats στην εφαρμογή μας.

- **com.google.gwt.xml.client**

  Αυτο το XML πακέτο δίνει την δυνατότητα στην εφαρμογή μας να κάνει parse σε ένα XML αρχείο και να το φορτώσει σε ένα DOM (Document Object Model). Υπάρχει και η δυνατότητα για το αντίστροφο, δηλάδή απο το DOM να πάμε στην δημιουργία ενός XML αρχείου. Η χρήση συτού του πακέτου μαζί με το HTTP έχει σαν αποτέλεσμα την ανταλλαγή XML μηνυμάτων ανάμεσα σε client-server.

- **com.google.gwt.json.client**

  Αυτό το πακέτο είναι παραπλήσιο με τον XML μόνο που το format των δεδομένων είναι JSON και όχι XML. Το JSON είναι παραπλήσιο με το XML μόνο που είναι πολύ πιο ελαφρύ και στέλνεται πολύ πιο γρήγορα μέσω

δικτύου. Με αυτό το πακέτο μπορούμε να δημιουργήσουμε και να κάνουμε parse JSON data.

Το JRE πακέτο περιέχει όλο το σύνολο του java runtime environment. Το GWT χρησιμοποιεί δύο απο τις πιο βασικές βιβλιοθήκες του JRE χωρίς να δημιουργεί καινούργιες για το βασικό προγραμματιστικό σκελετό όπως είναι exceptions, collection, Java types.

- **java.lang**

  Αυτό είναι το βασικότερο πακέτο της Java και περιέχει κλάσεις όπως Integer, String κλπ.

- **java.util**

  Το πακέτο utility μερικές απο τις πιο βασικές collection κλάσεις της Java όπως Maps, Lists Κλπ.

Η τελευταία κατηγορία πακέτων του GWT είναι τα utility πακέτα.

- **com.google.gwt.junit.client**

  Αυτό είναι ένα JUnit πακέτο για την δημιουργία κλάσεων που τεστάρουν τον κώδικα.

- **com.google.gwt.i18n.client**

  Το GWT υποστηρίζει internationalization μέσω αυτού του πακέτου.

- **com.google.gwt.core.ext**

  Αυτό το extension πακέτο παρέχει κλάσεις για να κάνουμε extend τον GWT compiler.

Αύτες είναι όλες οι κατηγορίες πακέτων που υπάρχουν. Δεν είναι υποχρεωτικό να χρησιμοποιηθούν όλες, τις περισσότερες φορές χρησιμοποιείται ένας συνδυασμός αυτών. Επίσης με το GWT υπάρχει η δυνατότητα χρήσης και third-party βιβλιοθηκών για την δημιουργία πιο ευέλικτης εφαρμογής.

4.3 Πλεονεκτήματα GWT

Το GWT είναι σχεδιασμένο με web tier αρχιτεκτονική που αυτομάτως το κάνει πολύ ευέλικτο. Μην ξεχνάμε γιατί οι χρήστε και οι developers έχουν αγκαλιάσει αυτή την αρχιτεκτονική ανάπτηξης λογισμικού, γιατι τους δίνει την δυνατότητα της κεντρικής διαχείρισης μέσω ενός συστήματος και δεν χρειάζεται να κάνει κάτι εγκατάσταση ο χρήστης τοπικά στο μήχανημα του. Η εφαρμογή μιλάει με τον browser. Ακόμη αν

συνδιαστεί και με το γεγονός ότι το GWT έχει βασιστεί πάνω στην τεχνολογία AJAX μπορούμε να πούμε ότι αυτόματα το καθιστούν ένα πολύ ανταγωνιστικό εργαλείο για συγγραφή εφαρμογών στο διαδίκτυο.

Άλλη μια σημαντική διαφορά του GWT η οποία το διαφοροποιεί απο άλλα RIA (Rich Internet Application) εργαλεία web development είναι δυνατότητα που δίνει για testing και debugging. Το GWT περιλαμβάνει ένα πανίσχυρο debugging shell για να κάνουμε debug στην εφαρμογή καθώς αλληλεπιδρά με τον browser. Ενω όσον αφορά το testing, το GWT μας παρέχει την δυνατότητα να χτίζουμε JUnit κλάσεις για να τεστάρουμε το κώδικα μας.

Τέλος όπως προαναφέρθηκε το GWT είναι cross-browser εργαλείο και οι εφαρμογές που γράφουμε με αυτό δεν εξαρτώνται απο τις ιδιοτροπίες κάθε browser. Επομένως η εμφάνιση των εφαρμογών είναι παντού ίδια ανεξαρτήτου λειτουργικού συστήματος και browser. Αυτο έιναι ένα ακόμη σημαντικό πλεονέκτημα το GWT για την δημιουργία web applications έναντι της παραδοσιακής δημιουργίας wen applications με HTML tags που είναι browser-dependent.

4.4 Μειονεκτήματα GWT

Πέρα απο τα σημαντικά πλεονεκτήματα το GWT έχει και μειονεκτήματα. Αρχικά ένα αρκετά immature εργαλείο το οποίο κυκολφόρησε μόλις το 2008 και έχει αρκετά Limitations μιας και βρίσκεται ακόμη σε αρχικό στάδιο.

Αρχικά πρέπει να επισημάνουμε την δυσκολία που έχουν οι μηχανές αναζήτησης στο web με την javascript. Τα περισσότερα search engine robots δεν μπορούν να πλοηγηθούν σε μία τέτοια εφαρμογή που δεν έχει απλά links για να πηγαίνεις απο τη μία σελίδα στην άλλη. Φυσικά αυτό δεν είναι μόνο πρόβλημα του GWT αλλά γενικότερα της AJAX τεχνολογίας στην οποία βασίζεται και το GWT. Βέβαια αυτό μπορεί να λυθεί απλά έχοντας μια version της εφαρμογής σε native-HTML η οποία θα γίνεται access μόνο απο τα search engine robots.

Ακόμη άλλο ένα μειωνέκτημα είναι ότι το GWT παρέχει έναν αρκετά καλά ορισμένο δικό του τρόπο για τον σχεδιασμό και την υλοποίηση της εφαρμογής που δεν προυποθέτει ότι πρέπει να τον γνωρίζεις. Δεν σου παρέχει απλά τα AJAX εργαλεία για να δουλέψεις και να σε καθιστά εσένα υπεύθυνο για την αρχιτεκτονική της εφαρμογής, αλλά θα πρέπει υποχρεωτικά να ακολουθήσεις την αρχιτεκτονική που σου παρέχει.

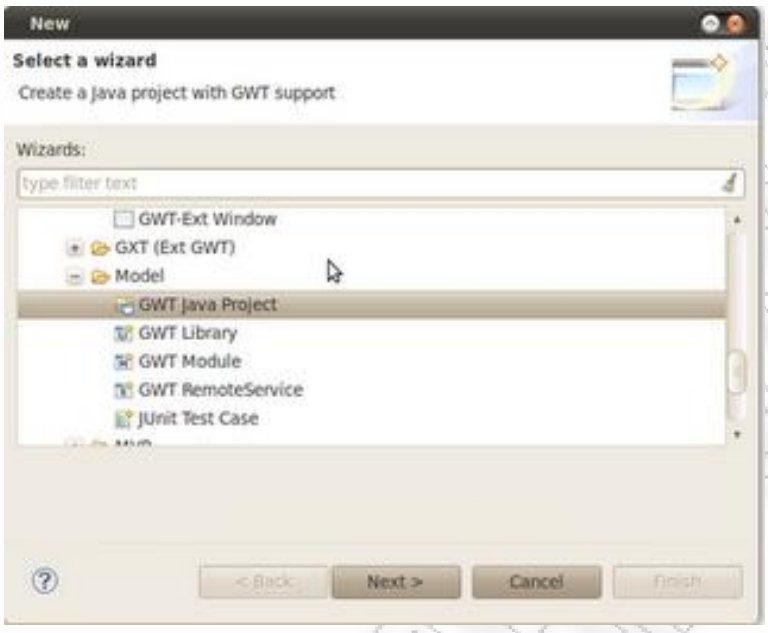
4.5 GWT στην πράξη

Παρακάτω θα δούμε ένα ολοκληρωμένο παράδειγμα δημιουργίας μιας εφαρμογής με GWT. Στα πλαίσια του παραδείγματος χρησιμοποιήσαμε Eclipse Helios IDE και gwt-

2.2. Και windobuilder ένα plugin του Eclipse για να φτιάχνουε έυκολα και γρήγορα φόρμες απλά κάνοντας drag n drop στην παλέτα σχεδιάσης. Στο παρακάτω παράδειγμα θα φτιάξουμε μια απλή εφαρμογή στην οποία θα στέλνουμε στον server με RPC το όνομα μας και θα απαντάει με πληροφοριές για το σύστημα μας ο server.

Αρχικά φτιάχνουμε ένα καινούριο gwt project.

New->Project->WindowBuilder->Model->GWT Java Project



Δίνουμε ένα όνομα στην εφαρμογή μας και μόλις πατήσουμε finish το project μας είναι έτοιμο και έχει φτιάξει τα παρακάτω modules και packages κάτω απο το src (Client,Server και Shared).Στο client πακέτο βάζουμε τον κώδικα που αναφέρεται στον client. Εκεί περα θα φτιάξουμε το UI της εφαρμογής και τα interfaces για τα services θα υλοποιήσουμε στο server πακέτο. Επομένως έχουμε τα παρακάτω interface

```
package com.cloudsms.client;

import com.google.gwt.user.client.rpc.RemoteService;

import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;


/**

 * The client side stub for the RPC service.

 */
```

```
@RemoteServiceRelativePath("greet")
public interface GreetingService extends RemoteService {
        String greetServer(String name) throws IllegalArgumentException;
}
```

Όπως είπαμε το παραπάνω αποτελεί το service που θα υλοποιήσουμε στο server πακέτο και ακολουθεί η υλοποίηση του στον server.

```
package com.cloudsms.server;

import com.cloudsms.client.GreetingService;
import com.cloudsms.shared.FieldVerifier;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

/**
 * The server side implementation of the RPC service.
 */
@SuppressWarnings("serial")
public class GreetingServiceImpl extends RemoteServiceServlet implements
                GreetingService {

        public String greetServer(String input) throws IllegalArgumentException {
                // Verify that the input is valid.
                if (!FieldVerifier.isValidName(input)) {
                // If the input is not valid, throw an IllegalArgumentException back to
                        // the client.
                        throw new IllegalArgumentException(
                                        "Name must be at least 4 characters long");
```

```
        }

String serverInfo = getServletContext().getServerInfo();
String userAgent = getThreadLocalRequest().getHeader("User-Agent");

        // Escape data from the client to avoid cross-site script vulnerabilities.
        input = escapeHtml(input);
        userAgent = escapeHtml(userAgent);

        return "Hello, " + input + "!<br><br>I am running " + serverInfo
                        + ".<br><br>It looks like you are using:<br>" +
                            userAgent;
}

/**
 * Escape an html string. Escaping data received from the client helps to
 * prevent cross-site script vulnerabilities.
 *
 * @param html the html string to escape
 * @return the escaped string
 */
private String escapeHtml(String html) {
        if (html == null) {
                return null;
        }
        return html.replaceAll("&", "&amp;").replaceAll("<", "&lt;")
```

```
                .replaceAll(">", "&gt;");
        }
}
```

Τέλος ακολουθεί η κεντρική κλάση που υπάρχει στο πακέτο του client και περιλαμβάνει όλο τον κώδικα για το UI καθώς και τον τρόπο με τον οποίο καλούμε το remote service στο event του button που στελνει το RPC αίτημα στον server.

```
package com.cloudsms.client;


import com.cloudsms.shared.FieldVerifier;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.event.dom.client.KeyCodes;
import com.google.gwt.event.dom.client.KeyUpEvent;
import com.google.gwt.event.dom.client.KeyUpHandler;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.DialogBox;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.VerticalPanel;
```

```
/**
 * Entry point classes define <code>onModuleLoad()</code>.
 */

public class CloudSms implements EntryPoint {
	/**
	 * The message displayed to the user when the server cannot be reached or
	 * returns an error.
	 */
	private static final String SERVER_ERROR = "An error occurred while "
			+ "attempting to contact the server. Please check your network"
			+ "connection and try again.";

	/**
	 * Create a remote service proxy to talk to the server-side Greeting service.
	 */
	private final GreetingServiceAsync greetingService =
GWT.create(GreetingService.class);

	/**
	 * This is the entry point method.
	 */
	public void onModuleLoad() {
		final Button sendButton = new Button("Send");
		final TextBox nameField = new TextBox();
		nameField.setText("GWT User");
		final Label errorLabel = new Label();
```

```
// We can add style names to widgets
sendButton.addStyleName("sendButton");

// Add the nameField and sendButton to the RootPanel
// Use RootPanel.get() to get the entire body element
RootPanel.get("nameFieldContainer").add(nameField);
RootPanel.get("sendButtonContainer").add(sendButton);
RootPanel.get("errorLabelContainer").add(errorLabel);

// Focus the cursor on the name field when the app loads
nameField.setFocus(true);
nameField.selectAll();

// Create the popup dialog box
final DialogBox dialogBox = new DialogBox();
dialogBox.setText("Remote Procedure Call");
dialogBox.setAnimationEnabled(true);
final Button closeButton = new Button("Close");
// We can set the id of a widget by accessing its Element
closeButton.getElement().setId("closeButton");
final Label textToServerLabel = new Label();
final HTML serverResponseLabel = new HTML();
VerticalPanel dialogVPanel = new VerticalPanel();
dialogVPanel.addStyleName("dialogVPanel");
dialogVPanel.add(new HTML("<b>Sending name to the
```

```
        server:</b>"));
        dialogVPanel.add(textToServerLabel);
        dialogVPanel.add(new HTML("<br><b>Server replies:</b>"));
        dialogVPanel.add(serverResponseLabel);

dialogVPanel.setHorizontalAlignment(VerticalPanel.ALIGN_RIGHT);
dialogVPanel.add(closeButton);
dialogBox.setWidget(dialogVPanel);

        // Add a handler to close the DialogBox
        closeButton.addClickHandler(new ClickHandler() {
                public void onClick(ClickEvent event) {
                        dialogBox.hide();
                        sendButton.setEnabled(true);
                        sendButton.setFocus(true);
                }
        });

        // Create a handler for the sendButton and nameField
        class MyHandler implements ClickHandler, KeyUpHandler {
                /**
                 * Fired when the user clicks on the sendButton.
                 */
                public void onClick(ClickEvent event) {
                        sendNameToServer();
                }
```

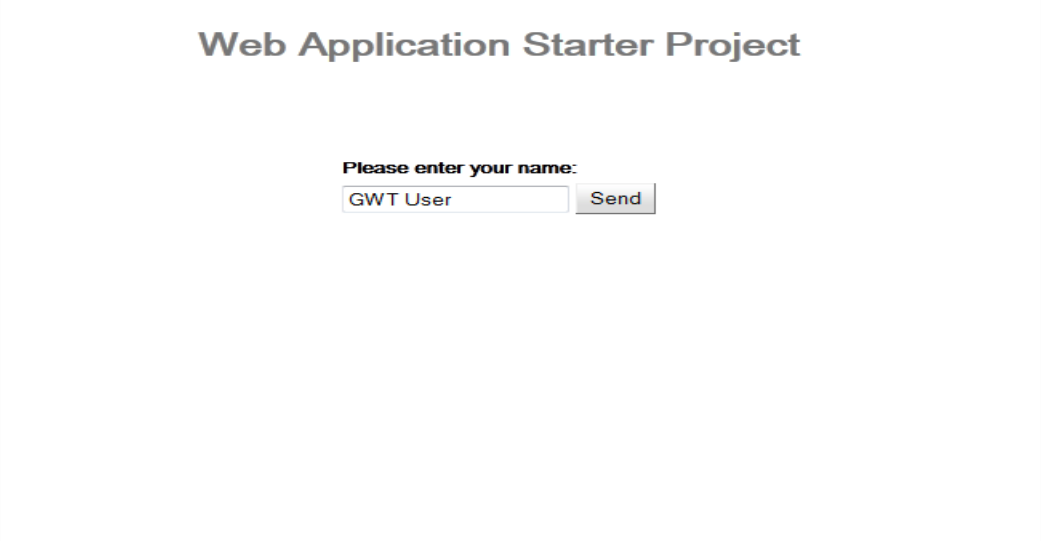
```
/**
 * Fired when the user types in the nameField.
 */
public void onKeyUp(KeyUpEvent event) {
if (event.getNativeKeyCode() == KeyCodes.KEY_ENTER) {
        sendNameToServer();
    }
}

/**
 * Send the name from the nameField to the server and wait for a response.
 */
private void sendNameToServer() {
    // First, we validate the input.
    errorLabel.setText("");
    String textToServer = nameField.getText();
    if (!FieldVerifier.isValidName(textToServer)) {
        errorLabel.setText("Please enter at least four
        characters");
        return;
    }

    // Then, we send the input to the server.
    sendButton.setEnabled(false);
    textToServerLabel.setText(textToServer);
    serverResponseLabel.setText("");
```

```
                                greetingService.greetServer(textToServer,
                                        new AsyncCallback<String>() {
                                    public void onFailure(Throwable caught) {
                            // Show the RPC error message to the user

        dialogBox.setText("Remote Procedure Call - Failure");
        serverResponseLabel.addStyleName("serverResponseLabelError");

        serverResponseLabel.setHTML(SERVER_ERROR);
        dialogBox.center();
        closeButton.setFocus(true);
}

public void onSuccess(String result) {

        dialogBox.setText("Remote Procedure Call");
        serverResponseLabel.removeStyleName("serverResponseLabelError");

        serverResponseLabel.setHTML(result);
        dialogBox.center();

        closeButton.setFocus(true);
}
});
}
}
```

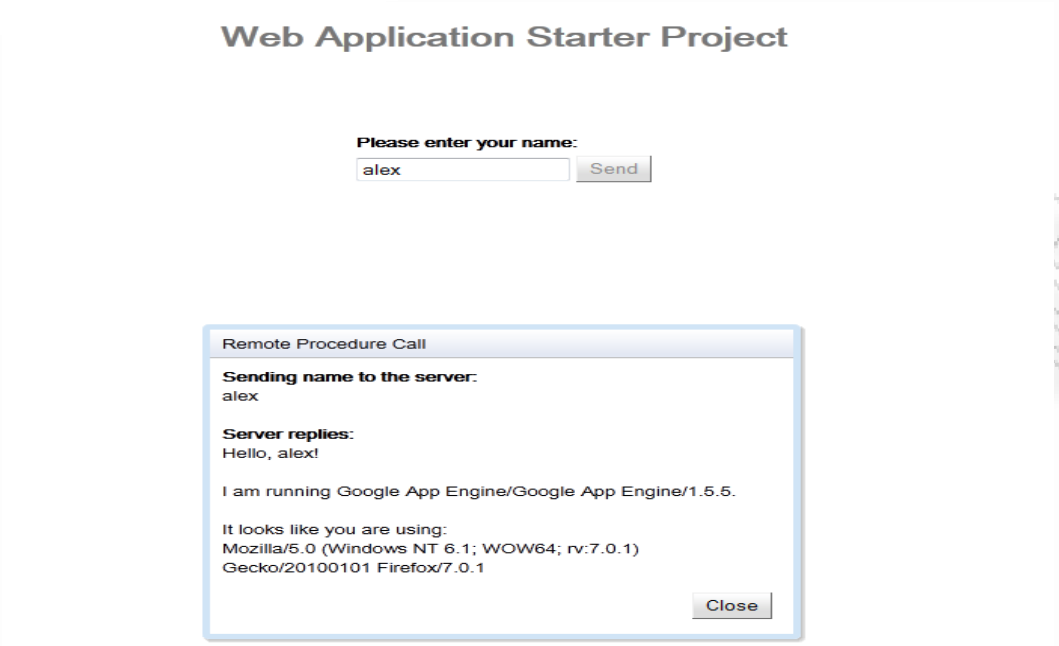
```
// Add a handler to send the name to the server

MyHandler handler = new MyHandler();

sendButton.addClickHandler(handler);

nameField.addKeyUpHandler(handler);

}
```

Τέλος κάνουμε compile στο project

δεξί κλικ στο Project -> Google -> GWT compile και ξεκινάει η διαδικασία του compilation. Μόλις τελειώσει μπορούμε να δούμε το αποτέλεσμα σε ένα web browser όπως φαίνεται παρακάτω.



Όταν πατήσουμε ένα όνομα και στείλουμε το αίτημα στον server μας απαντάει με το ακολουθο μήνυμα.



4.6 GWT Integration μαζί με JBoss Seam Framework

Παρακάτω θα παροσουσιάσουμε ένα παράδειγμα integration gwt με το JBoss Seam Framework. Το παράδειγμα αφορά Integration με το JBoss Seam και φαίνεται πολύ καθαρά πως μπορούμε να χρησιμοποιήσουμε τα πλούσια widgets που χρησιμοποιεί το GWT μαζί με την ωριμότητα που φέρνει ένα δοκιμασμένο framework για συγγραφή J2EE εφαρμογών που υποστηρίζει και persistency μέσω του JBoss Hibernate. Επομένως με το παρακάτω πράδειγμα έχουμε τον σκελετό για να χτίσουμε RIA (Rich Internet Applications) χρησιμοποιώντας τα ισχυρά και όμορφα widgets του GWT μαζί με όλες τις ευκολίες που μας δίνει το Seam για επικοινωνία με σχεσιακή βάση δεδομένων (π.χ Oracle) και όχι με το datastore.

Το παράδειγμα που ακολουθεί έχει υλοποιηθεί σε Eclipse Helios IDE και αποτελεί ένα απλό Login παράδειγμα που χρησιμοποιεί Oracle 10g βάση δεδομένων για να αποθηκεύσει τα στοιχεία των χρηστών, GWT για την δημιουργία του UI και την επικοινωνία με τον server και JBoss Hibernate για την επικοινωνία του server με την Βάση δεδομένων και το persistency με την βάση.

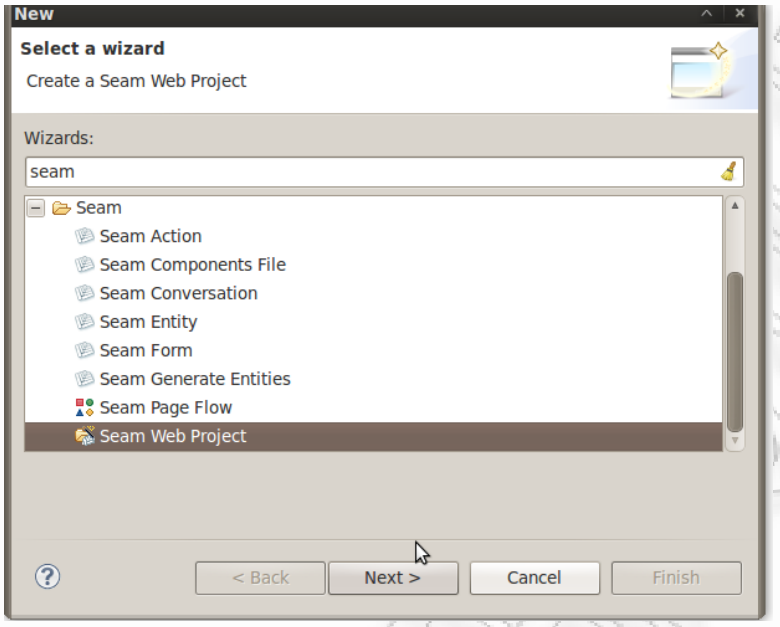
Αρχικά φτιάχνουμε έναν πίνακα σε oracle 10g για να αποθηκεύσουμε τα στοιχεία των χρηστών που θέλουμε.

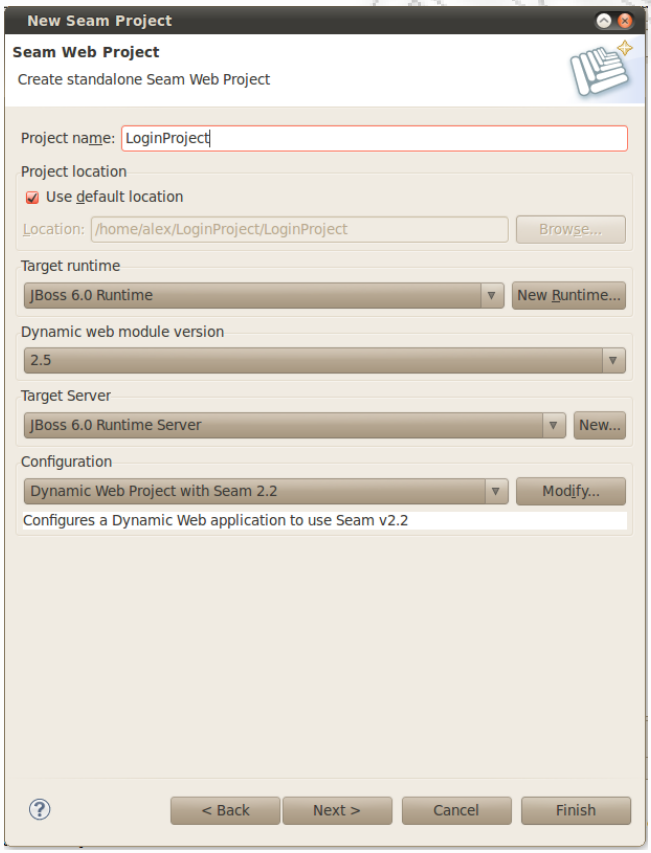
```
CREATE TABLE USER(

  Id serial NOT NULL PRIMARY KEY,

  Username varchar,
```

```
    Password varchar
);
```

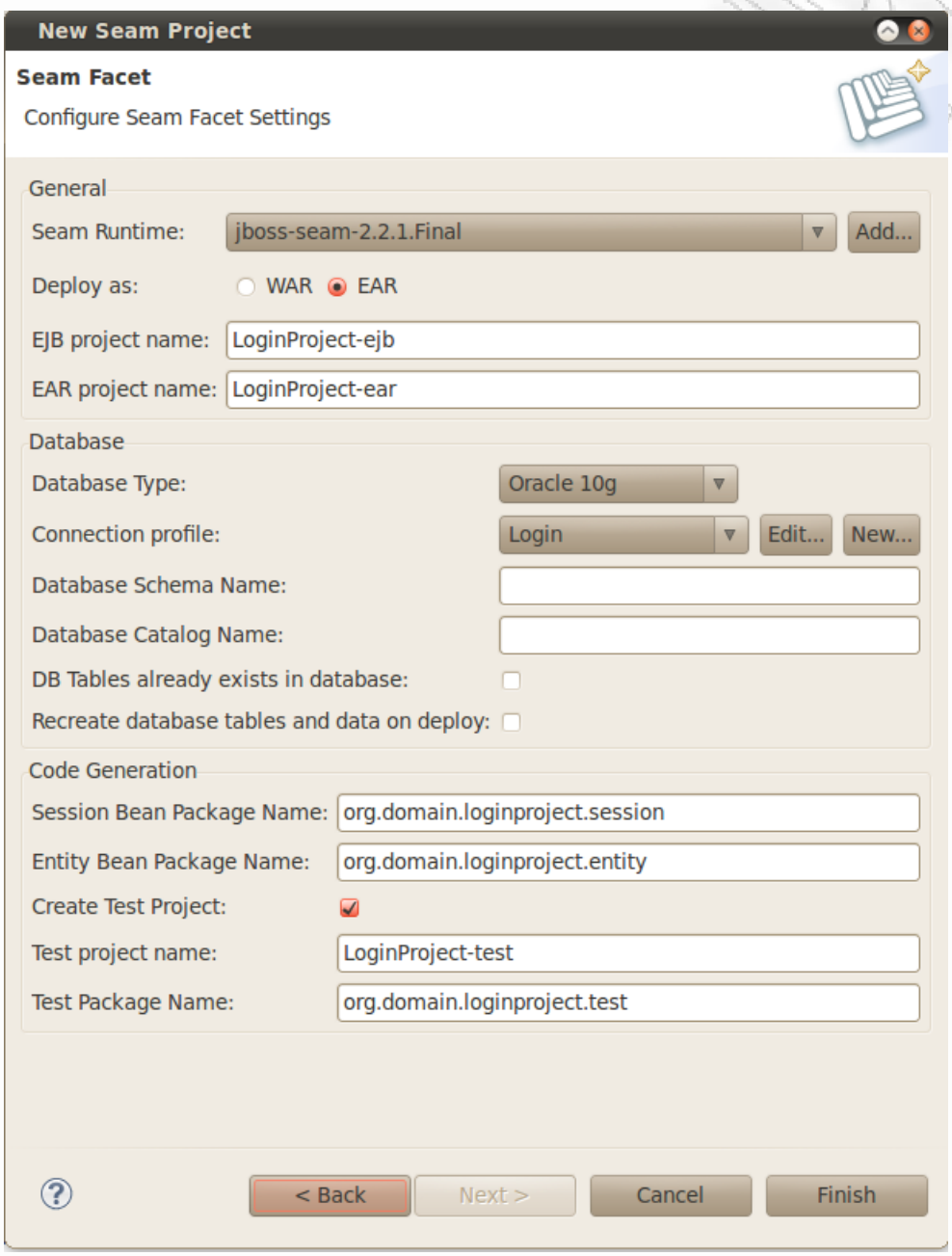
Στην συνέχεια και αφού έχουμε δημιουργήσει την βάση δεδομένων δημιουργούμε ένα απλό Seam Project ονόματι LoginProject στο eclipse όπως φαίνεται παρακάτω.

New->Project->Seam->Seam Web Project



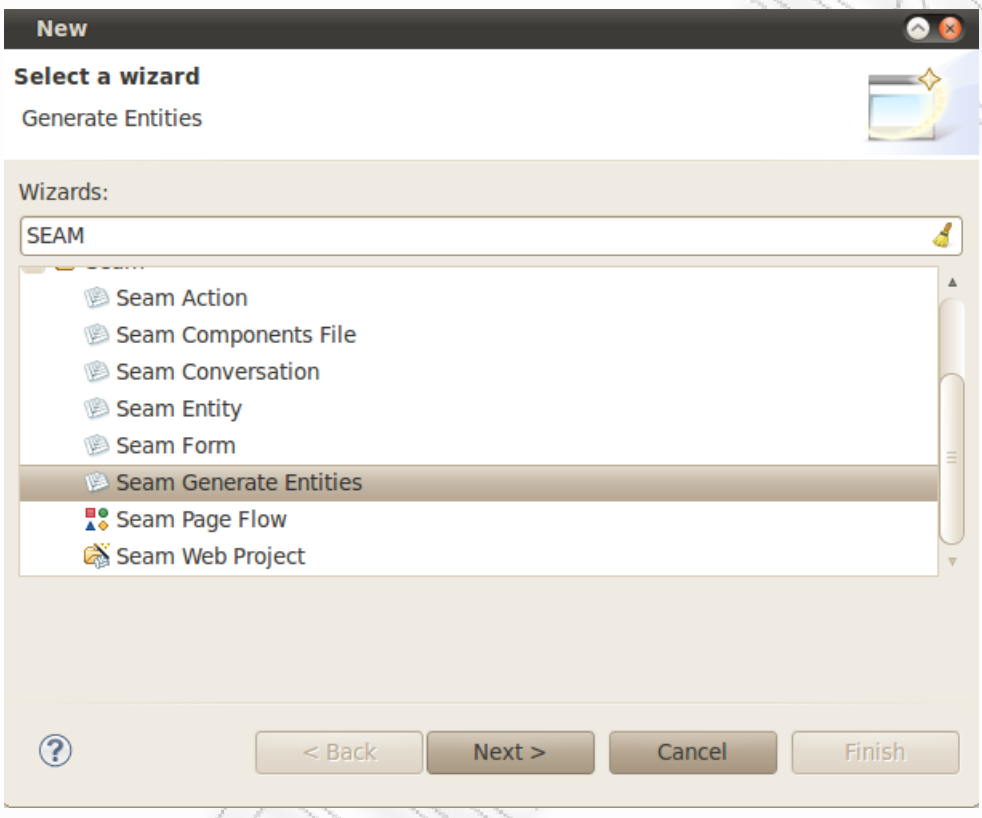


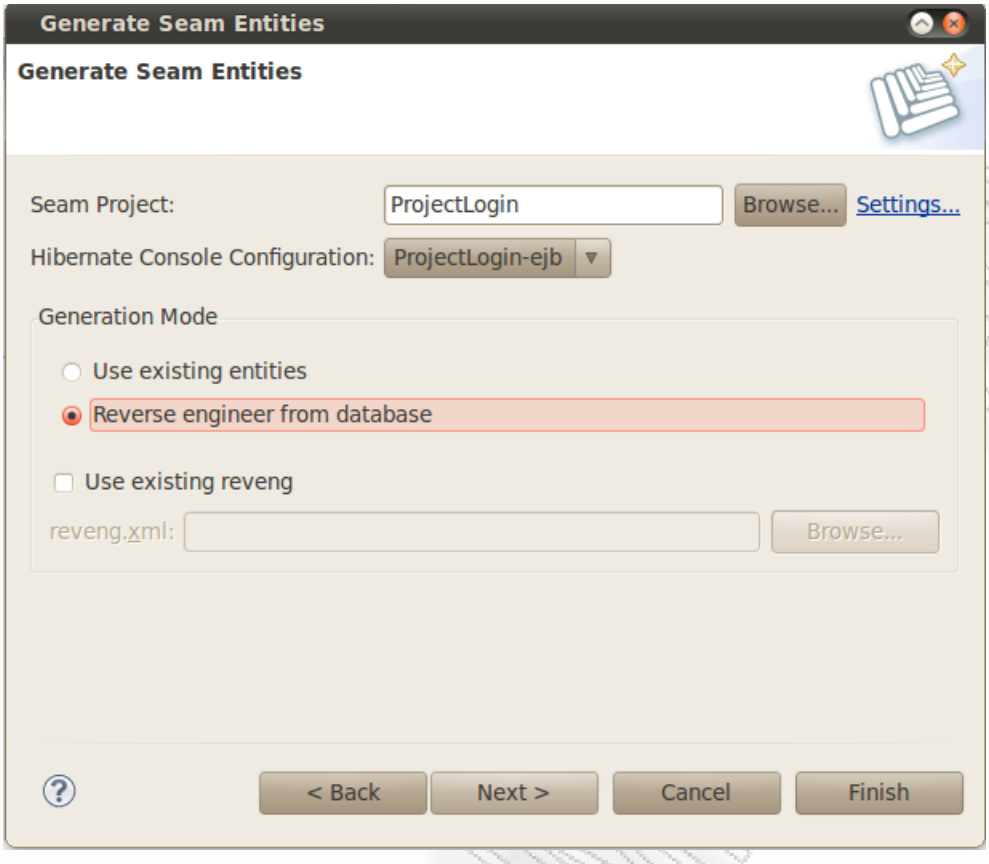
Στην συνέχεια και όπως φαίνεται παρακάτω πρέπει να δημιουργήσουμε το EJB module και το EAR module του project μας (LoginProject-ejb, LoginProject-ear) στα οποία Modules θα βρίσκονται στο μεν ejb (enterprise java beans) όλα java beans της εφαρμογής και στο δε ear θα βρίσκεται όλο το project που θέλουμε να γίνει deploy στον JBoss application server . Επίσης πρέπει να φτιάξουμε μια σύνδεση με την βάση δεδομένων μας για να μπορέσει να μιλήσει το persistency layer της java με την βάση δεδομένων που φτιάξαμε πριν.



Μετά την ολοκλήρωση αυτού του βήματος έχουν δημιουργηθεί τρία διαφορετικά Modules στην εφαρμογή μας το LoginProject, το LoginProject-ejb και το LoginProject-ear και έχει δημιουργηθεί έχει εγκαταστασθεί μία σύνδεση με την βάση δεδομένων. Στην συνέχεια πρέπει να δημιουργήσουμε τα entities που θα χρησιμοποιήσει η εφαρμογή μας απο την βάση δεδομένων που έχουμε ήδη φτιάξει. Αυτή η διαδικάσια ονομάζεται reverse engineer όπως βλέπουμε μας φτιάχνει entities απο τους πίνακες της βάσης δεδομένων.

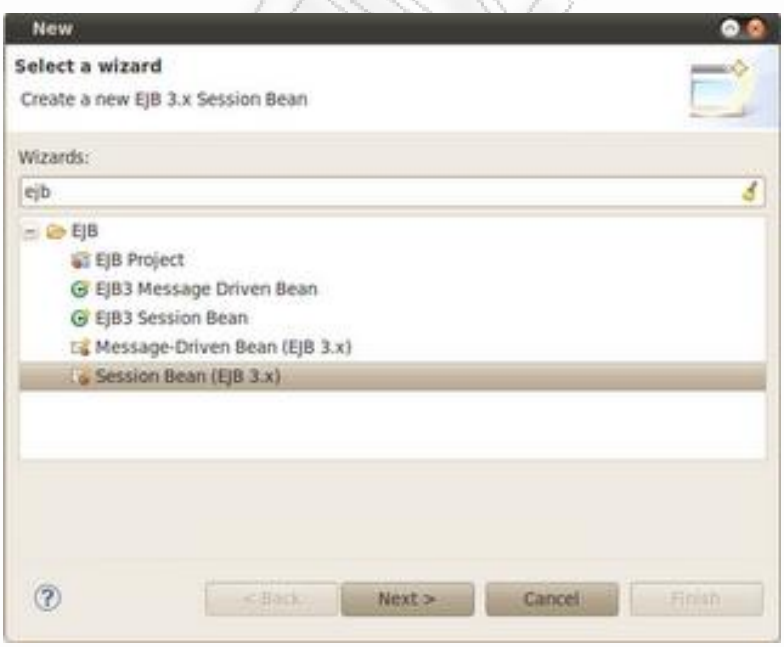
Δεξί κλικ στο EJB project → New → Project → Seam → Seam Generate Entities

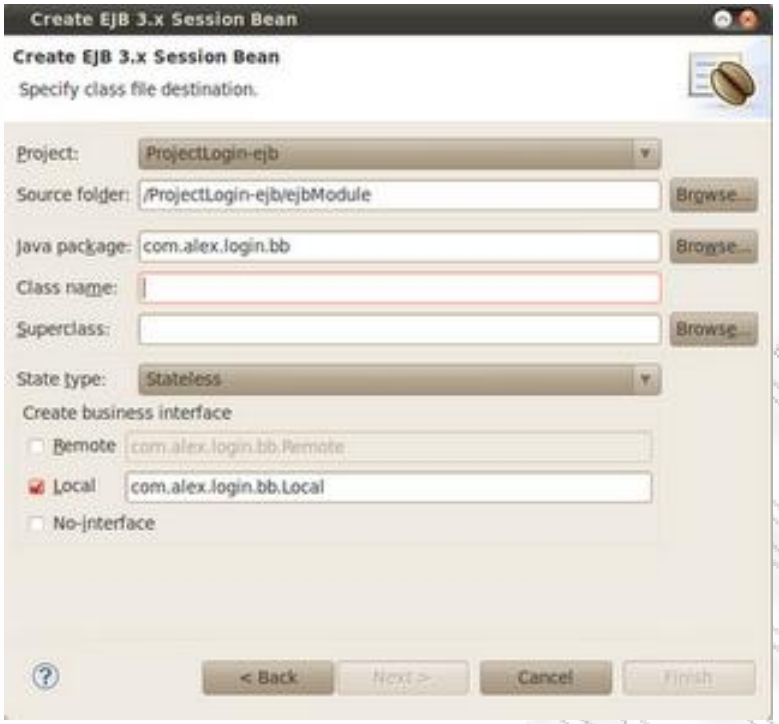




Μετά το τέλος αυτού του βήματος θα έχουν δημιουργηθεί τα entities μας στο ejb module. Το επόμενο βήμα που έχουμε να κάνουμε είναι να δημιουργήσουμε και κάποια session beans για να βάλουμε όλη τη λειτουργικότητα που θέλουμε να παρέχει η εφαρμογή μας.

Δεξί κλικ στο LoginProject-ejb → New → Project → EJB → Session Bean





Έτσι δημιουργείται ένα interface στο οποίο προσθέτουμε όλες τις μεθόδους που θέλουμε να υλοποιήσουμε και οι οποίες θα περιέχουν κάποια λειτουργικότητα για την εφαρμογή μας.

```
package com.alex.login.bb;
import javax.ejb.Local;

@Local
public interface authenticationLocal {
	public boolean checkUser(String username, String password);
}
```

Ακολουθεί και ο κώδικας της υλοποίησης αυτού του interface

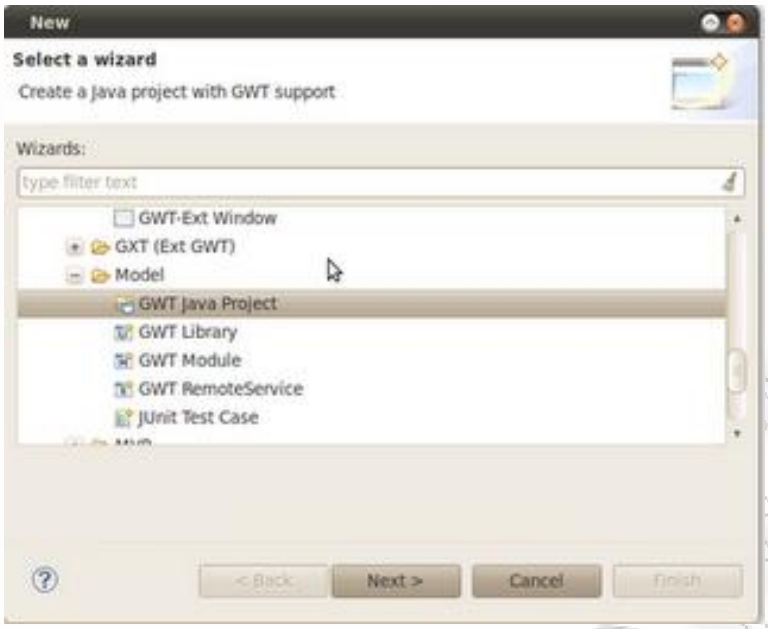
```
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

/**
* Session Bean implementation class authentication
*/
@Stateless
public class authentication implements authenticationLocal {

@PersistenceContext EntityManager em;
/**
* Default constructor.
*/
public authentication() {
```

```
// TODO Auto-generated constructor stub
}

public boolean checkUser(String username, String password) {
try {
      Query q = em.createQuery("Select U from Users U where
      UPPER(U.username) = UPPER(:username) AND UPPER(U.password) =
      UPPER(:password)");
      q.setParameter("username", username);
      q.setParameter("password", password);
try{
      q.getSingleResult();
}catch(Exception e){
      return false;
}
}catch (Exception e){
      return false;
}
return true;
}
}
```

Όπως φαίνεται στον παραπάνω κώδικα αυτό που κάνουμε είναι ένα απλό query στην βάση δεδομένων χρησιμοποιώντας το Hibernate και ελέγχουμε αν τα στοιχεία που εισήγαγε ο χρήστης (username, password) υπάρχουν στην βάση δεδομένων προκειμένου να τον αυθεντικοποιήσουμε.

Στην συνέχεια θα πάμε να δημιουργήσουμε ένα GWT Project και θα δούμε πως μπορούμε απο εκεί να πάμε να καλέσουμε το session bean που δημιουργήσαμε. Πρέπει να πούμε ότι έχουμε χρησιμοποιήσει ένα Plugin του Eclipse που ονάζεται WindowBuilder το οποίο μας δίνει την δυνατότητα να δημιουργούμε φόρμες με GWT widgets, απλά ζωγραφίζοντας στην παλέτα που μας δίνει και αυτό να γράφει απο πίσω μόνο του τον κώδικα.

New->Project->WindowBuilder->Model->GWT Java Project

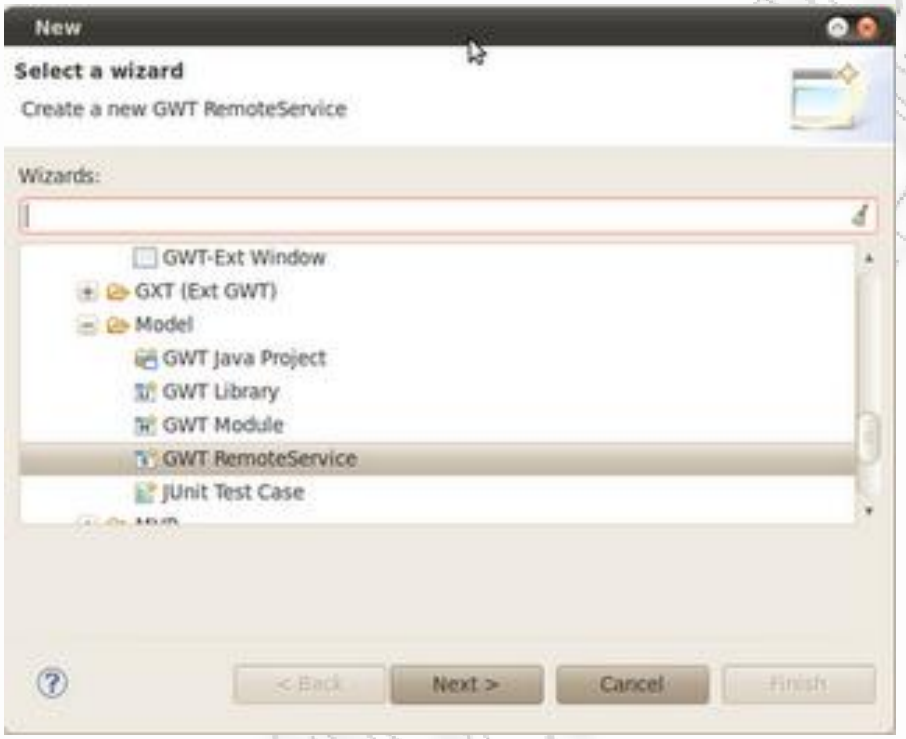


Πατώντας finish έχουμε φτιάξει ένα καινούριο Project που το ονομάζουμε LoginProject-gwt στο οποίο μέσα θα γράψουμε όλον το κώδικα για που αφορά το

GWT δηλαδή ότι αφορά UI και τον τρόπο με τον οποίο θα καλέσουμε το session bean μέσω RPC.

Όπως είπαμε αρχικά δημιουργούμε το UI μέσω του WindowBuilder απλά κάνωντας drag&drop τα widgets του GWT που θέλουμε. Το σημαντικό κομμάτι της εφαρμογής είναι το πως θα δημιουργήσουμε ένα Remote Service το οποίο θα καλεί μέσω RPC το session bean.

Δεξί κλικ στο LoginProject-gwt→New → WindowBuilder → Model → GWT Remote Service



Απλά δίνουμε ένα όνομα στο Remote Service που θέλουμε να φτιάξουμε και πατάμε finish. Ακολουθεί ο κώδικας του Remote Service

LoginService.java

```
package com.alex.login.client;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.RemoteService;
import
com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("LoginService")
public interface LoginService extends RemoteService {
     public boolean checkLogin(String username, String
password);
}
```

Παρακάτω ακολουθεί και η υλοποίηση του interface σε κώδικα.

LoginServiceImpl.java

```
package com.alex.login.server;

import javax.ejb.EJB;

import com.alex.login.bb.authenticationLocal;
import com.alex.login.client.LoginService;
import
com.google.gwt.user.server.rpc.RemoteServiceServlet;

public class LoginServiceImpl extends
RemoteServiceServlet implements LoginService {

@EJB authenticationLocal auth;
@Override
public boolean checkLogin(String username, String
password) {
return auth.checkUser(username, password);
}
}
```

Τέλος καλούμε το το παραπάνω remote service απο το event του button του UI. Όπως φαίνεται παρακάτω στι γραμμές που είναι bold..

```
package com.alex.login.client;

import com.google.gwt.core.client.EntryPoint;

/**
 * Entry point classes define <code>onModuleLoad()</code>.
 */
public class Login implements EntryPoint {
        private Button btnLogin;

    private final LoginServiceAsync loginService =
GWT.create(LoginService.class);

        public void onModuleLoad() {
            RootPanel rootPanel = RootPanel.get();

      ContentPanel cntntpnlLogin = new ContentPanel();
                    cntntpnlLogin.setHeading("Login");
                   cntntpnlLogin.setCollapsible(true);
            cntntpnlLogin.setLayout(new AbsoluteLayout());
```

```
                        btnLogin      =      new      Button();
      cntntpnlLogin.add(btnLogin, new AbsoluteData(111, 119));
                  btnLogin.setSize("66px",              "23px");
                                      btnLogin.setText("Login");

         final   TextField   txtUsername   =   new   TextField();
        cntntpnlLogin.add(txtUsername,   new   AbsoluteData(77,
21));

        final   TextField   txtPassword   =   new   TextField();
        cntntpnlLogin.add(txtPassword,   new   AbsoluteData(77,
68));
                                   txtPassword.setFieldLabel("");

      LabelField lblfldUsername = new LabelField("Username:");
       cntntpnlLogin.add(lblfldUsername,  new  AbsoluteData(6,
26));

      LabelField lblfldPassword = new LabelField("Password:");
       cntntpnlLogin.add(lblfldPassword,  new  AbsoluteData(6,
73));
          btnLogin.addClickHandler(new       ClickHandler(){
         public     void     onClick(ClickEvent     event)     {

loginService.authenticateUser(txtUsername.getValue().toString(
), txtPassword.getValue().toString(), new
AsyncCallback<Boolean>()                                    {

                                                      @Override
        public     void     onSuccess(Boolean     result)     {
                                    if                (result){
                  Window.alert("Successfully   logged   in!");
                                                      }else{
                              Window.alert("Unknown      user.");
                                                      }
                                                      }

                                                      @Override
        public     void    onFailure(Throwable     caught)     {
                                    Window.alert("Problem.");
                                                      }
                                                     });

                                                      }
  });
                                  rootPanel.add(cntntpnlLogin);
          rootPanel.setWidgetPosition(cntntpnlLogin,  32,  47);
                  cntntpnlLogin.setSize("300px",      "200px");
                                                      }
}
```

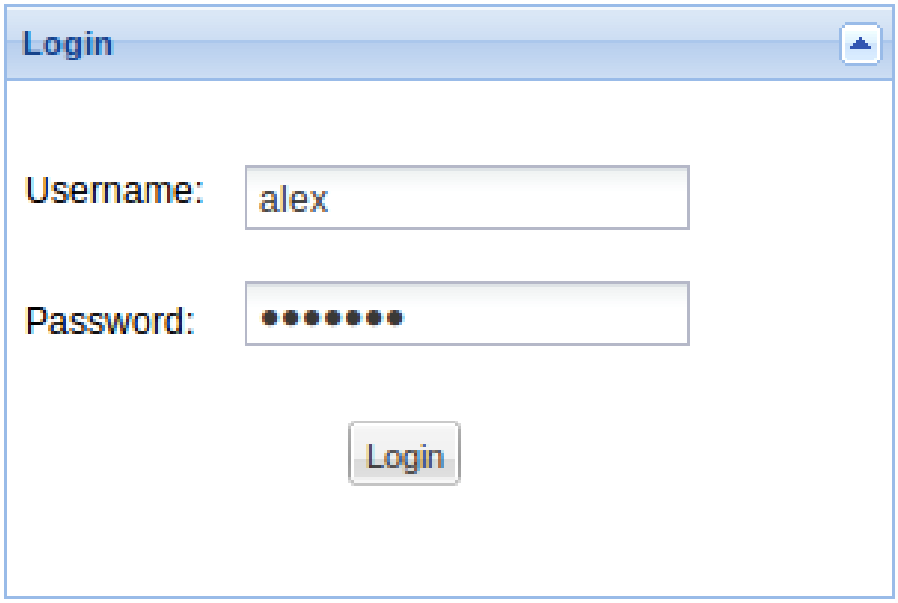
Ο παραπάνω κώδικας είναι κατα ένα μεγαλο βαθμό παράγωγο της δημιουργίας της φόρμας μας απο το Windowbuilder όταν ζωγραφίσαμε στην παλέτα. Το μόνο το οποίο αρκούσε να κάνουμε εμείς είναι να βάλουμε κώδικα στο event του Login Button. Αφού τελειώσουμε και αυτό κάνουμε compile σε όλο το gwt project που δημιουργήσαμε και είμαστε έτοιμοι να τρέξουμε την εφαρμογή μας

Το επόμενο βήμα που έχουμε να κάνουμε είναι κάποια έξτρα παραμετροποίηση στο gwt project που δημιουργήσαμε για να μπορεί να μιλήσει με τα άλλα project.
Δεξί κλικ στο LoginProject-gwt→Properties→Project facets→ τικάρουμε το Dynamic Web Module and make the appropriate configurations

Στην συνέχεια ένα πολύ σημαντικό βήμα που έχουμε να κάνουμε είναι να κάνουμε το EAR που θα φορτώσουμε στον application server να μπορεί να δεί το GWT Project .
Δεξί κλικ στο LoginProject-ear → Deployment Assembly → Add → LoginProject-gwt

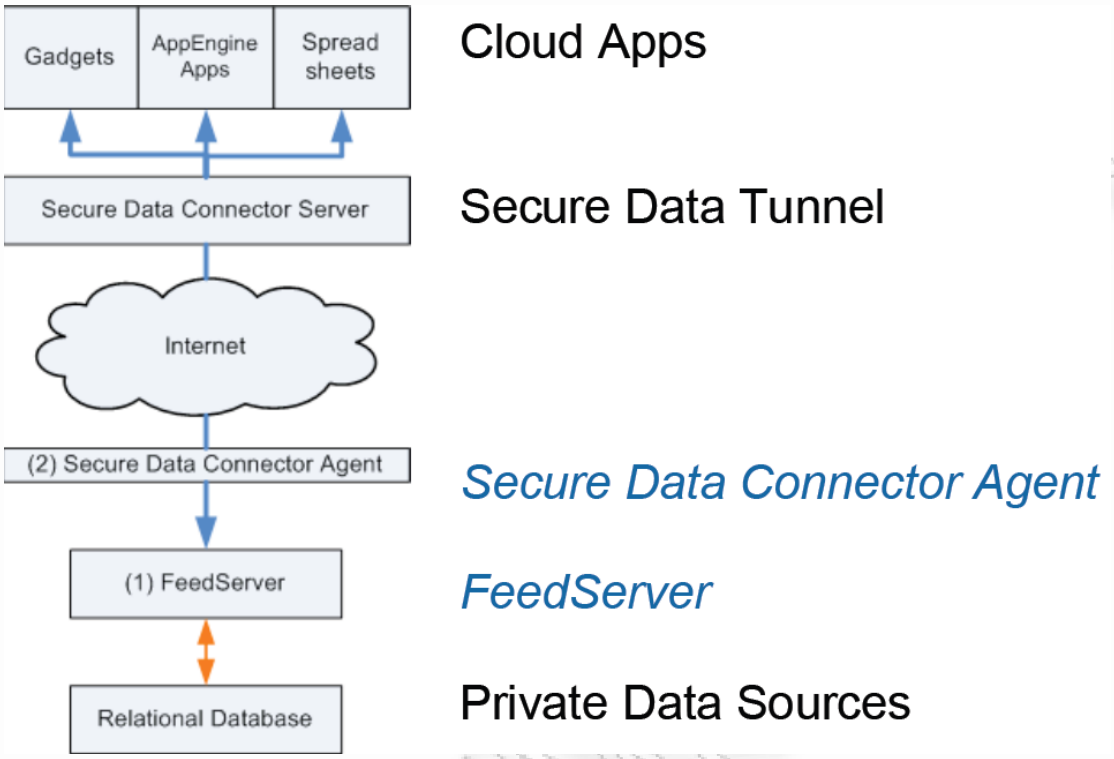
Και τέλος φορτώνουμε το ear στον application server (JBoss στην περίπτωση μας).
Δεξί κλικ στο server tab του Eclipse → Add/Remove → επιλέγουμε το ear που θέλουμε.

Τέλος όπως βλέπουμε και παρακάτω όταν τρέχουμε την εφαρμογή στον browser μας εμφανίζεται η παρακάτω GWT φόρμα η οποία καλέι ένα remote service το οποίο ένα session που μιλάει με την βάση δεδομένων και μας κάνει αυθεντικοποίηση.



5. Secure Data Connector

Το secure Data Connector αποτελεί ένα εργαλείο της Google για να μπορεί να μιλάει απευθείας με βάσεις δεδομένων άλλων συστημάτων. Παρακάτω βλέπουμε την βασική αρχιτεκτονική του.



(Πηγή: «Google Installing and configuring Secure Data Connector presentation», Jun Yang, Google).

Η βασική ιδέα για την δημιουργία προέκυψε απο το γεγονός ότι υπάρχουν πολλοί προγραμματιστές οι οποίοι θέλουν έχουν σε δικία τους βάση δεδομένων όλα τα δεδομένα τους και όχι στο datastore της google επομένωνς θα έπρεπε να βρεθεί ένας τρόπος να μπορεί η εφαρμογή να επικοινωνήσει με την βάση και να πάρει ότι δεδομένα χρειάζεται με ασφαλή τρόπο.

Όπως φαίνεται και παραπάνω απο την αρχιτεκτονική του Data Connector έχουμε απο την πλευρά της βάσης δεδομένων το Secure Data Connector Agent και απο την πλευρά της εφαρμογής μας το Secure Data Connector Server και αυτά τα δύο συστατικά μέρη επικοινωνούν μέσω Internet μέσω ενός secure tunnel που φτιάχνουν μεταξύ τους.

Απο την πλευρά της βάσης το μόνο που χρειάζεται είναι ένας Feedserver ο οποίος μετατρέπει τα relational δεδομένα που έχει μια παραδοσιακή relational βάση δεδομένων σε Atom Feeds που μπορεί να καταλάβει μια εφαρμογή σε GWT. Έτσι αφού γίνει ο μεταχηματισμός ο client του Secure Data Connector επιστρέφει μέσω Internet αυυτά τα δεδομένα στο Server και ο μετά ο Server στην εφαρμογή μας.

5.1 Secure Data Connector στην πράξη

Αρχικά θα δημιουργήσουμε την βάση δεδομένων μας και τον πινακα το οποίο θέλουμε να χρησιμοποιήσουμε.

```
CREATE TABLE employee (
 id  NOT NULL PRIMARY KEY,
 lastname varchar,
 firstname varchar,
 jobtitle varchar,
 location varchar,
 phoneoffice varchar);
```

Αφού δημιουργήσουμε το πίνακα κατεβάζουμε τον Google Feedserver ο οποίος όπως είπαμε θα μετασχηματίσει τα δεδομένα σε Atom Feeds. Στην παρακάτω διεύθυνση http://code.google.com/p/google-feedserver/.

Μετά πρέπει να τροποποιήσουμε το αρχείο conf/database/dbConfig.properties ως εξής

```
JDBC.Driver = <driver βάσης που χρεισημοποιούμε>
JDBC.ConnectionURL = jdbc:<βάση δεδομένων που χρησιμοποιούμε>:<URL βάσης δεδομένων>
JDBC.Username = <username>
JDBC.Password = <password>
JDBC.Jar = <jar αρχείο για τον driver της βάσης δεδομένων>
```

Αυτά είναι τα βασικά properties τα οποία πρέπει να αλλάξουμε για να μπορέσει ο Feedserver να συνδεθεί με την βάση δεδομένων μας. Στην συνέχεια πρέπει να ορίσουμε πως θα γίνει η μετατροπή των δεδομένων των δεδομένων για κάθε μια απο τις πιθανές αναζητήσεις που μπορεί να κάνει ο χρήστης. Επομένως πάμε και φτιάχνουμε ένα αρχείο στο `/conf/feedserver/resources/FeedConfig/employeeSearch.xml`

Το οποίο θα μας γυρίζει θα μας γυρίζει το όνομα και την τοποθεσία στην οποία βρίσκεται ο εργαζόμενος

```
<select id="employeeSearch-get-feed" resultMap="result-map.xml">
     select * from employee
     where firstname like #name#
     and location like #location#
     order by firstname
     limit 20 offset 0
</select>
```

Επομένως όταν ψάνχουμε για έναν εργαζόμενο με βάση το όνομα και την τοποθεσία (Location) στην οποία βρίσκεται θα μας γυρίζει τα στοιχεία του με βάση το ακόλουθο mapping αρχείο .

```
/conf/feedserver/resources/FeedConfig/result-map.xml
```

```
<resultMap id="employee" class="HashMap">
 <result property="id" column="id"/>
 <result property="lastName" column="lastname"/>
 <result property="firstName" column="firstname"/>
 <result property="jobTitle" column="jobtitle"/>
 <result property="location" column="location"/>
 <result property="phoneOffice" column="phoneoffice"/>
</resultMap>
```

Αυτό το αρχείο πάει και κάνει mapping στην ουσία τα πεδία του Atom Feed με τις αντιστοιχες στήλες στον πίνακα της βάσης δεδομένων απο όπου τραβάει τα δεδομένα.

Απο την πλευρά της βάσης δεδομένων είμαστε έτοιμοι καθώς ο μετασχηματισμός δεδομένων έχει γίνει και το μόνο που χρειάζεται είναι να κάνουμε εγκατάσταση το Secure Data Connector για να μπορέσει να μεταφέρει τα δεδομένα μέσω internet στην εφαρμογή που είναι στο Google App Engine.

Αρχικά κατεβάζουμε το Secure data Connector Agent (http://code.google.com/sdc) το οποίο το κάνουμε εγκατάσταση στο μηχάνημα όπου βρίσκεται και η βάση δεδομένων μας και ορίζουμε το URL της εφαρμογής όπου θέλουμε να επιστρέψει τα δεδομένα. Τροποποιούμε το connector.xml που βρίσκεται στον φάκελο του Secure data connector που κατεβάσαμε.

```
<url>[URL_ΕΦΑΡΜΟΓΗΣ]</url>
```

Τέλος τρέχουμε το secure data connector τρέχοντας το start.sh αρχείο που υπάρχει στο φάκελο εγκατάστασης που κατεβάσαμε νωρίτερα και όλα είναι έτοιμα απο την πλευρά της βάσης δεδομένων. Τώρα το μόνο που απομένει είναι να δούμε πως στέλνουμε αίτημα απο κάποια εφαρμογή που βρίσκεται στο Google App Engine για να ζητήσει δεδομένα.

Απο την πλευρά του Server το μόνο που αρκεί να κάνουμε είναι να φτιάξουμε ένα bean στο οποίο θα αποθηκεύονται τα δεδομένα που θέλουμε, και να δούμε πως στέλνουμε αίτημα στον agent για να μας επιστέψει αυτά τα δεδομένα. Επομένως για το συγκεκριμένο παράδειγμα φτιάχνουμε το ακόλουθο bean

```
Pubic class Employee {
Private String lastName;
Private String firstname;
Private String jobTitle;
Private String location;
Private String phoneOffice;
Public String getLastName (){
     Return lastName;
}
Public void setLastName (String ln){
     this.lastName = ln;
}
Public String getFirstName (){
     Return lastName;
}
Public void setFirstName (String fn){
     this.firstName = fn;
}
```

```
Public String getJobTitle (){
     Return jobTitle;
}
Public void setJobTitle (String jt){
     this.jobTitle = jt;
}
Public String getLocation (){
     Return location;
}
Public void setLocation (String lo){
     this.location = lo;
}
Public String getPhoneOffice (){
     Return phoneOffice;
}
Public void setPhoneOffice (String po){
     this.phoneOffice = po;
}
}
```

Στην συνέχεια βλέπουμε πως μπορούμε να στείλουμε ένα αίτημα στον Secure Data Connector Agent απο την Google App Engine εφαρμογή για να μας στείλει δεδομένα απο την βάση δεδομένων.

```
GoogleService service = new GoogleService();
FeedServerClient <Employee> client =
new FeedServerClient<Employee>(service, Employee.class);
service.getRequestFactory();
URL feedUrl = new URL("THE_URL_OF_OUR_APPLICATION");
List<Employee> employees = client.getEntities(feedUrl);
```

Έτσι με το παραπάνω κομμάτι κώδικα μπορούμε να φτιάξουμε service το οποίο σαν σκοπό θα έχει να ζητήσει δεδομένα απο την εξωτερική βάση δεδομένων και να της τα στείλει πίσω το Secure Data Connector.

6. Προτάσεις χρήσης – Συμπεράσματα

Σαν συμπέρασμα μπορούμε να πούμε ότι οι παραπάνω τεχνολογίες βοηθούν πολύ στην δημιουργία RIA εφαρμογών (Rich Internet Applications) που θέλουμε να βρίσκονται στην υποδομή της Google. Τι γίνεται όμως σε περίπτωση που θέλουμε απλά να χρησιμοποιήσουμε όλες τις δυνατότητες που μας δίνει το Google Cloud (Google AppEngine, GWT, κλπ.) χωρίς να θέλουμε να ανεβάσουμε τα δεδομένα μας στο cloud της. Σε αυτή την περίπτωση όπως δείξαμε και παραπάνω μπορούμε να επικοινωνήσουμε με κάποια εξωτερική βάση δεδομένων και όχι με το Datastore, που μας παρέχει η Google, και να έχουμε εμείς όπου θέλουμε τα δεδομένα μας καθώς επίσης να είμαστε και εμείς υπεύθυνοι για την ασφάλεια τους. Αυτή η δυνατότητα γίνεται μέσω του Secure Data Connector ενός εργαλείου το οποίο δίνει την δυνατότητα σε εφαρμογές που βρίσκονται στο Cloud της Google να επικοινωνούν με εξωτερικά συστήμα βάσεων δεδομένων μέσω ένος secure tunnel μέσω internet.

Επίσης μπορούμε να χρησιμοποίησουμε μόνο το GWT για τα πολύ εύχρηστα και όπτικα όμορφα widgets που μας παρέχει αλλά και τον ταχύτατο τρόπο που επικοινωνεί με το Server. Ακολουθώντας το παράδειγμα που αναφέρεται στο κεφάλαιο 4.6 το GWT μπορεί πολύ εύκολα να επικοινωνήσει με κάποιο άλλο framework για ανάπτυξη εφαρμογών σε J2EE (Java 2 Enterprise Edition) όπως στο παράδειγμα με το JBoss Seam. Με αυτό το τρόπο κερδίζουμε το ότι επικοινωνούμε με μία relational βάση δεδομένων, μέσω hibernate στην προκειμένη περίπτωση, και όχι object βάση δεδομένων όπως είναι το Datastore. Έτσι έχουμε τη δυνατότητα να μιλάμε με οποιαδήποτε βάση θέλουμε η οποία μπορεί να βρίσκεται οπουδήποτε εκτός της υποδομής του Google και να έχουμε έμείς τον έλεγχο και την ευθύνη για την ασφάλεια των δεδομένων μας.

Επομένως ένα γενικό συμπέρασμα θα μπορούσαμε να πούμε ότι είναι μπορούμε να χτίσουμε πραγματικά γρήγορες, λειτουργικές και πολύ φιλικές προς τον χρήστη εφαρμογές χρησιμοποιώντας το Google Web Toolkit (πολύ έξυπνα και οπτικά όμορφα AJAX widgets και ταχύτατη επικοινωνία με τον server μέσω RPC). Επίσης μέσω του Google AppEngine μπορούμε να διαχειριστούμε εύκολα και γρήγορα την εφαρμογή που βρίσκεται στην υποδομή της Google μέσω της πληθώρας δυνατοτήτων που μας δίνει το GAE (page view statistics, logs, cron tasks, κλπ.). Τέλος ένα απο τα σημαντικότερα εργαλεία που έχει κατασκευάσει η Google για να μπορεί να επικοινωνεί και με άλλα συστήματα εκτός του Cloud της είναι το Secure Data Connector το οποίο είναι ενας μετατροπέας των δεδομένων που πέρνονται απο το

εξωτερικό σύστημα σε Atom Feeds (GData) το οποίο είναι το πρότυπο των δεδομένων που μπορεί να καταλάβει μια εφαρμογή στο Google Cloud.

Βιβλιογραφία

Βιβλία

- «Pro Web 2.0 Application Development with GWT May 2008», *Jeff Dywer*, εκδόσεις Apress publishing.
- «Google Web Toolkit 2 Application Development Cookbook», *Shamsuddin Ahammad*, Packt publishing.
- «Google Web Toolkit Applications», *Ryan Dewsbury*, Prentice Hall.
- «Manning GWT in Practice», *Robert T. Cooper*, *Charlie E. Collins*, Manning publications.
- «Programming Google App Engine», *Dan Sanderson*, O'Reilly
- «Using Google App Engine», *Charles Severance*, O'Reilly
- «Google Installing and configuring Secure Data Connector presentation», Jun Yang, Google

Ηλεκτρονικές διευθύνσεις

- http://code.google.com/intl/el-GR/appengine/
- http://code.google.com/intl/el-GR/webtoolkit/
- http://en.wikipedia.org/wiki/Google_App_Engine
- http://en.wikipedia.org/wiki/Google_Web_Toolkit
- http://googlewebtoolkit.blogspot.com/
- http://code.google.com/intl/el-GR/appengine/articles/datastore/overview.html
- http://www.ctoedge.com/content/how-google-app-engine-datastore-works
- http://www.databasejournal.com/features/mssql/article.php/3823471/Cloud-Computing-with-Google-DataStore.htm
- http://code.google.com/intl/el-GR/securedataconnector/
- http://www.google.com/support/a/bin/answer.py?answer=115739