# Προηγμένα Θέματα Λογισμικού Υπηρεσιών Ιστού

**Event Sourcing Pattern**

**Command Query Responsibility Segregation (CQRS)**

## Τι είναι το Event Sourcing Design Pattern

Το Event sourcing είναι ένας όρος Design Pattern και ανήκει στην κατηγορία Αρχιτεκτονικού μοτίβου ( Architectural Pattern). Εάν κάτι έχει συμβεί ή συμβαίνει , δηλαδή ένα συμβάν, είναι υπεύθυνο για την αποθήκευση του συμβάντος σε ένα μέρος (για παράδειγμα σε έναν πίνακα σε βάση δεδομένων), στην συγκεκριμένη στιγμή που έγινε.

# Γενική ιδέα για το Event Souring:

- Διασφάλιση ότι κάθε αλλαγή στην κατάσταση μιας εφαρμογής καταγράφεται σε ένα αντικείμενο συμβάντος (Object Event), και αυτά τα αντικείμενα γεγονότων αποθηκεύονται από μόνα τους στην σειρά που εφαρμόστηκαν.
- Η απλή αναφορά πηγών προέλευσης περιέχει ένα αρχείο καταγραφής (Log File) που εφαρμόζονται σε ένα αντικείμενο (Object)
- Η καταγραφή αυτών των μεταγενέστερων αλλαγών μπορεί να γίνει χρήσιμη όταν θέλουμε να επαναλάβουμε τις αλλαγές που έγιναν στα αντικείμενα μας (για οποιοδήποτε λόγο).

## Event sourcing έχει τα παρακάτω πλεονεκτήματα:

- Είναι ικανό να αποθηκεύει όλες τις καταστάσεις μιας αλλαγής.
- Παρέχει μια 100% αξιόπιστη καταγραφή ελέγχου (audit log) των αλλαγών που έγιναν σε μια επιχειρηματική οντότητα
- Κάνει δυνατή την εφαρμογή χρονικών ερωτημάτων που καθορίζουν την κατάσταση μιας οντότητας ανά πάσα στιγμή.
- Από τη στιγμή που κρατάει όλα τα συμβάντα και τις χρονικές στιγμές που έγιναν , είναι εύκολο να επαναφέρουμε μια κατάσταση σε προηγούμενη ( Rollback)

# Event sourcing έχει τα παρακάτω μειονεκτήματα:

- Πρόκειται για ένα διαφορετικό και άγνωστο στυλ προγραμματισμού και έτσι υπάρχει μια καμπύλη μάθησης.
- Η αποθήκευση των Events (Event Store) είναι δύσκολο να επεκταθεί, καθώς απαιτεί ερωτήματα (queries) για την ανακατασκευή της κατάστασης των επιχειρηματικών οντοτήτων. Ως αποτέλεσμα, η εφαρμογή πρέπει να χρησιμοποιεί ένα άλλο Design Pattern που είναι γνωστό ως Command Query Responsibility Segregation (CQRS) , το οποίο είναι υπεύθυνο για την υλοποίηση των ερωτημάτων (Queries).

# Σχετικά μοτίβα ( που συνδυάζονται)

- Τα πρότυπα (Patterns ) Saga και  Domain Event δημιούργησαν την ανάγκη για το Event Sourcing Pattern.
- Το CQRS πρέπει συνήθως ως πάντα να χρησιμοποιείτε σε συνδυασμό με το event sourcing.
- Event sourcing υλοποιεί το πρότυπο καταγραφής ελέγχου του λογαριασμού (Audit logging Pattern).

# Τι είναι το CQRS

CQRS είναι η ακρωνύμια του Command Query Responsibility Segregation. Από το ίδιο το όνομα του Design Patterns καταλαβαίνουμε ότι είναι υπεύθυνο αυτό το Design Pattern για 2 εργασίες που αναλαμβάνει. Το Command και το Query. Το command είναι υπεύθυνο να εμφανίζονται τυχόν αλλαγές. Το Query είναι υπεύθυνο να συγκεντρώσει τα δεδομένα σύμφωνα με τη ανάγκη της εφαρμογής που θα υλοποιηθεί. Με απλά λόγια ουσιαστικά το Command και το Query είναι να διαβάζουν και να γράφουν (Read & Write). Το CQRS βοηθάει τον προγραμματιστή της εφαρμογής να σχεδιάσει την δομή της εφαρμογής όπου η ανάγνωση και η γραφή διαχωρίζονται. Κάνει την εφαρμογή ασφαλέστερη και μεγιστοποιεί την απόδοση της.

# Πλεονεκτήματα του CQRS

- Υποστηρίζει πολλαπλές απομνημονευμένες προβολές που είναι επεκτάσιμες και αποδοτικές
- Βελτιωμένος διαχωρισμός των ανησυχιών = απλούστερα μοντέλα εντολών και ερωτήσεων
- Απαραίτητο σε μια αρχιτεκτονική που προέρχεται από συμβάντα (event sourcing)

# Μειονεκτήματα του CQRS

- Αυξάνει την πολυπλοκότητα
- Πιθανό επαναλαμβανόμενος – αντιγραφή (Duplication) κώδικα
- Μεσοπρόθεσμη αναπαραγωγή / τελικά συνεπείς προβολές

# Σχετικά μοτίβα ( που συνδυάζονται)

- The Database per Service pattern δημιουργεί την ανάγκη για τη χρήση αυτού του Pattern
- Το Api Composition Pattern είναι μια εναλλακτική λύση
- Το Domain Event Pattern δημιουργεί τα συμβάντα (events)
- Το CQRS συνήθως χρησιμοποιείτε με το Event Sourcing Pattern

# Γιατί ο προγραμματιστής χρησιμοποιεί ή δεν χρησιμοποιεί Event Sourcing

Το Event Sourcing δεν είναι κάτι που μπορεί να χρησιμοποιηθεί σε κάθε εφαρμογή. Ένα από τα πιο δύσκολα κομμάτια είναι να καταλάβεις που να το χρησιμοποιήσεις. Κάνει την εφαρμογή ανιχνεύσιμη και ασφαλής. Αν κάπως χάσεις δεδομένα ή κάτι συμβεί απροσδόκητα, μπορείς να το ανακτήσεις από το συμβάν, δηλαδή μπορεί να εντοπιστεί εύκολα και να λυθεί. Δεδομένου ότι η ανάγνωση δεδομένων έχει προτεραιότητα όταν τα συμβάντα μεταφέρονται στη βάση δεδομένων μέσω query, γίνεται εύκολη και φορτώνει λιγότερο για να διαβάσει τα δεδομένα, γεγονός που καθιστά την εφαρμογή πιο γρήγορη και αποδοτική. Το Event Sourcing είναι καλό για ένα μεγάλο και σύνθετο Project όπου μπορεί να χρειαστεί καταγραφή ιστορικών γεγονότων. Η χρήση του Event Sourcing σε μικρή ή απλή εφαρμογή CRUD θα καταστήσει πολύπλοκότητα χωρίς να δίνει σημαντικά οφέλη

# Γιατί ο προγραμματιστής χρησιμοποιεί ή δεν χρησιμοποιεί CQRS

Είναι ένα από τα πιο έντονα μοτίβα σχεδιασμού, αν χρησιμοποιηθεί σωστά. Κάνει να διαβάσει και να γράψει, ανεξάρτητα του φόρτου εργασίας του Server και να κρατήσει το κώδικα καθαρό. Όταν το Event Sourcing απαιτεί περίπλοκο ερώτημα, το CQRS το καθιστά απλό. Ο προγραμματιστής εφαρμόζει το CQRS επίσης για να χειριστεί πάρα πολλά αιτήματα από την Βάση δεδομένων.

# Υλοποιημένο παράδειγμα στην Εφαρμογή

Η εφαρμογή που θα παρουσιαστεί έχει να κάνει με καταθέσεις και μεταφορές μετάξύ τραπεζικών λογαριασμών. Υπάρχουν 2 είδη χρηστών στην εφαρμογή, ο απλός χρήστης και ο Manager.

Ο απλός χρήστης έχει σαν επιλογή να καταθέσει χρήματα καθώς θα είναι και σε θέση να μεταφέρει χρήματα στο λογαριασμό άλλου χρήστη.

Ο Manager μπορεί να διαγράψει το ποσό που κατέβαλε ο χρήστης και να διακόψει την συναλλαγή του χρήστη. Επίσης μπορεί να βλέπει μια λίστα με όλους τους χρήστες και τα ποσά που έχει ο κάθε ένας από αυτούς.

# Γιατί χρησιμοποιήθηκε το Event Sourcing & CQRS

- Για την παρακολούθηση της κατάθεσης και της μεταβίβασης μεταξύ των Λογαριασμών.
- Για να ελαχιστοποιηθεί η πιθανότητα σύνθετων Query με την χρήση όπως Joins.
- Για την διατήρηση καθαρού κώδικα
- Για εύκολη επέκταση – προσθήκη δυνατότητας

# Δομή της βάσης

Η Δομή της βάσης αποτελείτε από 4 πίνακες

- Users(αποθηκεύει τα στοιχεία των απλών Χρηστών)
- Admins (αποθηκεύει τα στοιχεία των Managers Χρηστών)
- Deposits ( Χρησιμοποιείτε για τα ποσά των χρηστών)
- Event_stores (Χρησιμοποιείτε για να αποθηκεύει τα events)

# Δραστηριότητα απλών Χρηστών

## Κατάθεση

- Show Amount – Το αίτημα πηγαίνει στην μέθοδο του Controller του Account. Εκεί γίνετε η αρχικοποίηση του QueryService ανάλογα το e-mail του χρήστη και σαν αποτέλεσμα παίρνει το υπολειπόμενο ποσό.
- Deposit – Τα αιτήματα των χρηστών τους ανακατευθύνουν στο Account Controller. Για να καταθέσουν υπάρχει μια μέθοδος που καλείτε η μέθοδος AccountCommandHandler για να χειριστεί την εντολή ( η εντολή αυτή είναι ένα event). Περνούν παράμετρο με το e-mail του χρήστη και το ποσό που έχει τοποθετήσει. Σε αυτή τη κλάση το event δημιουργεί μια κλήση στο CreateDepositEvent με JSON μορφή τα δεδομένα.
- Υπάρχουν ακόμα 2 κλάσεις που είναι το ένα το EventStore και το άλλο είναι το EventHandler. Το πρώτο είναι υπεύθυνο για την αποθήκευση του Event στον πίνακα evet_stores με εντολή. Το EventHandler χρησιμοποιείτε για τη διαχείριση του event, όπου τα δεδομένα των καταθέσεων πηγαίνουν στο πίνακα deposits.

# Δραστηριότητα απλών Χρηστών

## Μεταφορά

Στο περίπου η μεταφορά είναι όπως και η κατάθεση.

Το αίτημα μεταφοράς πηγαίνει στη μέθοδο AccountController. Εδώ κάνει αίτηση για τα δεδομένα (αποστολέας,δέκτης και το ποσό) έχουν εισαχθεί στο αντικείμενο AccountTransferCommand το οποίο περνάει τα στοιχεία στο AccountTransferHandler.

Στο AccountTransferHandler, δημιουργείτε ένα event transfer μέσω του CreateTransferEvent. Το EventStore δημιουργεί το event. Και στη συνέχει πάει στο EventHandler. Στην εκτέλεση του EventHandler εκτελέστηκε με επιτυχία αν το ποσό είναι μικρότερο ή ίσο από το συνολικό ποσό του αποστολέα, διαφορετικά αποτύχασε η αποστολή.

# Δραστηριότητα Manager Data Reading

- Λίστα με τις καταθέσεις των χρηστών

όταν εκτελεστεί το αίτημα έρχετε το AccountList του ManagerController, που η μέθοδος getAccountList του AdminQueryService καλείτε

- Ιστορικό

Κάθε event εμφανίζεται εδώ. Το Αίτημα του ιστορικού πρώτα έρχετε στη μέθοδο history της κλάσης του Manager. Στην συνέχεια πάει στο getHistory του AdminQueryService

# Δραστηριότητα Manager Data Writing

- Διακοπή μεταφοράς

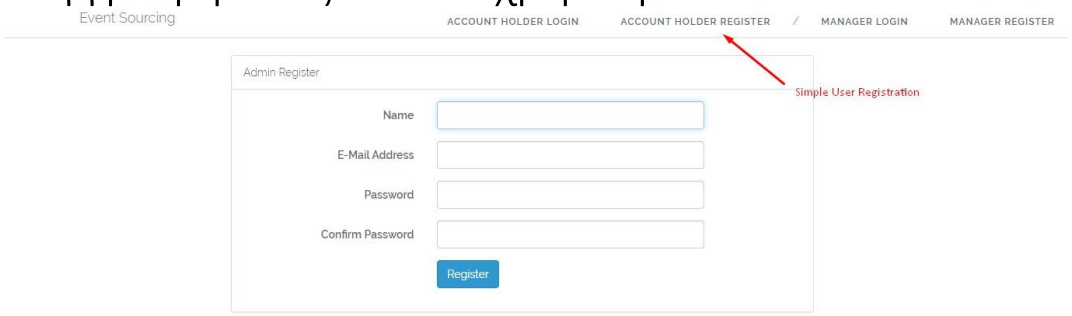
Το αίτημα έρχεται από τη σελίδα του ιστορικού. Πρώτα πηγαίνει στη μέθοδο stopTransaction της κλάσης Manager και δημιουργεί ένα object του AccoundStopCommand αρχικοποιώντας τον αποστολέα, παραλήπτη και το ποσό , και περνάει το object αυτό στο AccountStopCommandHandler για τη δημιουργία, αποθήκευση και διαχείριση του event. Στον EventHandler ο αποστολέας παίρνει το μεταφερόμενο ποσό και ο παραλήπτης χάνει το ποσό από το Συνολικό του.

- Διαγραφή Μεταφοράς

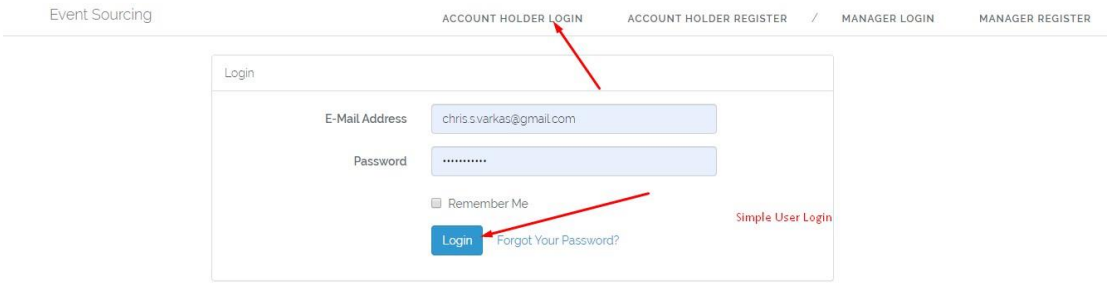
Για την διαγραφή της μεταφοράς υπάρχει η μέθοδος του Manager δημιουργεί ένα Object AccountDeleteCommand και ΄το στέλνει στο AccountDeleteCommandHandler. Πρώτα το event δημιουργείτε και το event αποθηκεύετε και στο τέλος το event διαχειρίζεται στη διαγραφή της συγκεκριμένης εγγραφής του πίνακα deposits.

## Παρουσίαση παραδείγματος:

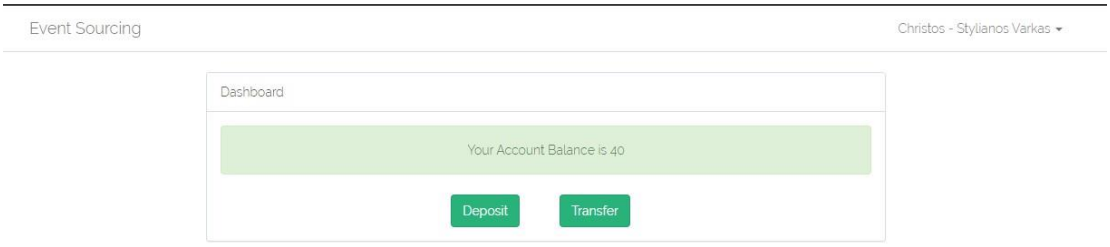
## 1 Εγγραφή ενός απλού χρήστη



# Σύνδεση απλού Χρήστη

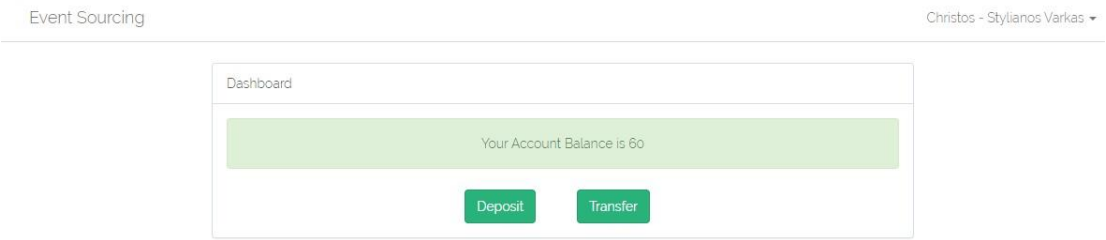



# Dasboard 1ου απλού χρήστη



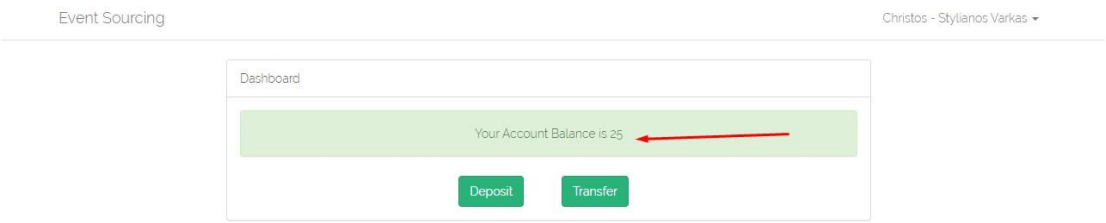
Κάνουμε κατάθεση άλλα 20 ευρώ άρα με τα 40 που είχε πριν θα πάει στα 60


Εδώ βλέπουμε ότι έχει 60 και στη συνέχεια μεταφέρει ένα ποσό 35 ευρώ σε άλλον απλό χρήστη λογαριασμό

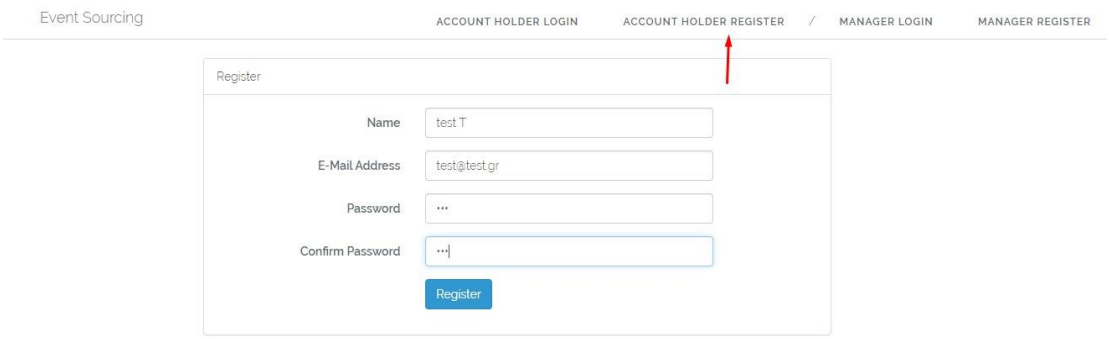


# Βάζουμε το ποσό μεταφοράς και τον παραλήπτη


# Βλέπουμε τη μείωση του ποσού στον χρήστη

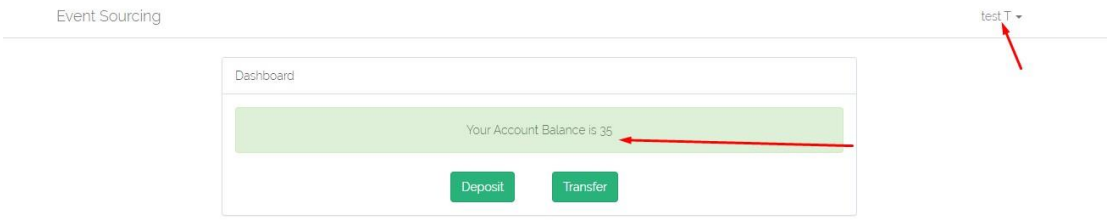


Εδώ φτιάχνουμε 2ο απλό χρήστη (Και να μην υπήρχε πριν ο χρήστης, το e-mail παραλήπτη που βάλαμε πριν) όταν δημιουργηθεί του βάζει το ποσό που του είχε γίνει αποστολή

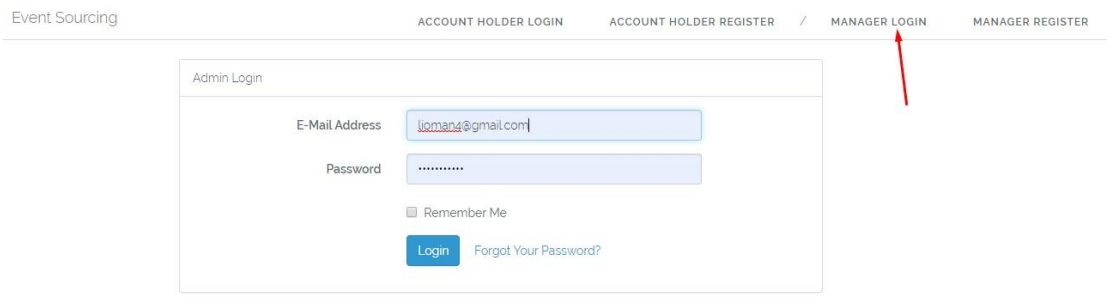



# Συνδεόμαστε με τα στοιχεία του δεύτερου απλού χρήστη

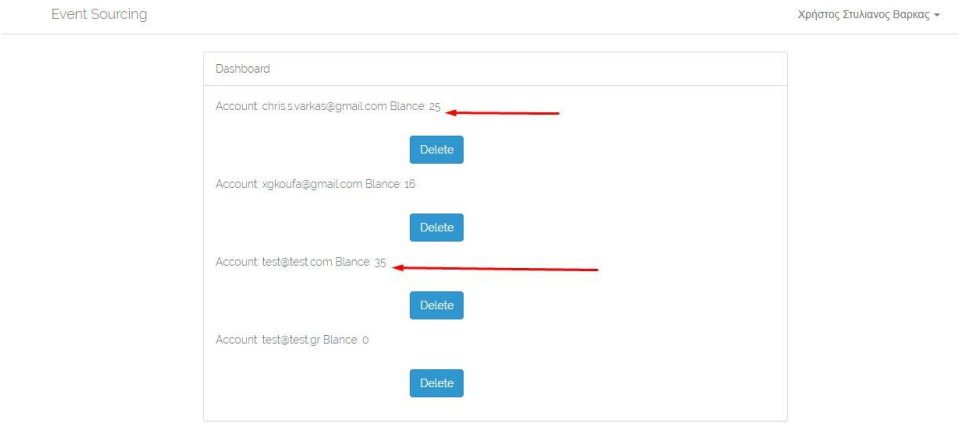
Βλέπουμε ότι το ποσό ενώ έπρεπε να είναι 0 έχει τα 35 που του έστειλε ο πρώτος χρήστης πριν την δημιουργία του




Συνδεόμαστε σαν Manager της εφαρμογής



Εδώ στο Dashboard βλέπουμε όλους τους χρήστες του συστήματος και το ποσό που έχει ο κάθε λογαριασμός




Επίσης βλέπουμε μια καταγραφή που κρατάει με όλα τα Commands που έγιναν και με ποια σειρά