* Graphical user interface

  Description automatically generated with medium confidence

Λογισμικό υπηρεσιών ιστού με χρήση προτύπων σχεδίασης

Εργασια για το μαθημα: Προηγμένα Θέματα Λογισμικού Υπηρεσιών Ιστού

Ονοματεπώνυμο: Τζεμπελίκου Σοφία

ΑΜ: mpsp19049

e-mail: [mpsp19049@unipi.gr](mailto:mpsp19049@unipi.gr)

Καθηγητής: Επικ. Καθ. Ευάγγελος Σακκόπουλος

Table of Contents

[Λογισμικό υπηρεσιών ιστού με χρήση προτύπων σχεδίασης 0](file:////Users/sophia/Documents/GitHub/CloudDesignPatternsUnified/Λογισμικό%20υπηρεσιών%20ιστού%20με%20χρήση%20προτύπων%20σχεδίασης.docx#_Toc83384943)

[Πρόλογος 2](#_Toc83384944)

[Τα design patterns που επιλέχθηκαν (Θεωρητικό Υπόβαθρο) 3](#_Toc83384945)

[CQRS Pattern 3](#_Toc83384946)

[Event Sourcing Pattern 5](#_Toc83384947)

[Retry pattern 8](#_Toc83384948)

[Δομή του Project 10](#_Toc83384949)

[Δομή του CQRS Pattern 10](#_Toc83384950)

[Δομή του EventSourcing Pattern 15](#_Toc83384951)

[Δομή του Retry Pattern 17](#_Toc83384952)

[Δομή του MainApp project και επεξήγηση της ενοποίησης των τριών design patterns. 19](#_Toc83384953)

[Class Diagram 23](#_Toc83384954)

[Πηγές 23](#_Toc83384955)

# 

# Πρόλογος

Στο software engineering, μετά από χρόνια χρήσης του κώδικα από τους προγραμματιστές, έγινε φανερό ότι κάποια προβλήματα στη συγγραφή του κώδικά ήταν συχνά επαναλαμβανόμενα. Έτσι με το πέρας του χρόνου ήταν φανερό ότι για να εξαλειφθούν τα προβλήματα αυτά ήταν αναγκαίο να βρεθούν κοινές λύσεις οι οποίες θα έλυναν τα προβλήματα και το development θα γινόταν ευκολότερο. Δημιουργήθηκαν λοιπόν τα πρότυπα σχεδίασης. Το κάθε πρότυπο σχεδίασης ή αλλιώς design pattern είναι μία επαναχρησιμοποιούμενη λύση σε ένα επαναλαμβανόμενο πρόβλημα. Δεν είναι ένα τελειωμένο σχέδιο που μπορεί να μετατραπεί απευθείας σε κώδικα μηχανής. Μάλλον, είναι μια περιγραφή για τον τρόπο επίλυσης ενός προβλήματος που μπορεί να χρησιμοποιηθεί σε πολλές διαφορετικές καταστάσεις.

Τα πρότυπα σχεδίασης είναι επίσημες βέλτιστες πρακτικές που μπορεί να χρησιμοποιήσει ο προγραμματιστής για την επίλυση κοινών προβλημάτων κατά το σχεδιασμό μιας εφαρμογής ή ενός συστήματος. Τα cloud design patterns είναι επίσης πρότυπα σχεδίασης κώδικα τα οποία παρέχουν λύσεις για κοινά προβλήματα που παρουσιάζονται κατά την ανάπτυξη cloud-hosted εφαρμογών. Στη παρούσα εργασία θα παρουσιαστεί ένα ενδεικτικό παράδειγμα τριών cloud design patterns τα οποία λειτουργούν συνδυαστικά σε μία ενιαία λύση. Η εφαρμογή αυτή είναι μία console εφαρμογή και αναπαριστά μια δανειστική βιβλιοθήκη.

# Τα design patterns που επιλέχθηκαν (Θεωρητικό Υπόβαθρο)

Στην εργασία αυτή έχει υλοποιηθεί μια εφαρμογή κονσόλας η οποία αναπαριστά μια δανειστική βιβλιοθήκη. Για την υλοποίηση έχουν επιλεγεί τα παρακάτω design patterns:

## CQRS Pattern

Σε μία συνηθισμένη CRUD αρχιτεκτονική, το ίδιο μοντέλο δεδομένων χρησιμοποιείται τόσο για την αναζήτηση των δεδομένων στη βάση όσο και για την ενημέρωση της βάσης δεδομένων. Αυτή η προσέγγιση είναι απλή και λειτουργεί πολύ σωστά σε βασικές λειτουργείες CRUD (Create, Read, Update, Delete). Σε πιο σύνθετες εφαρμογές η τακτική αυτή μπορεί να γίνει δύσχρηστη καθώς κατά τη διαδικασία αναζήτησης και ενημέρωσης της βάσης δεδομένων, υπάρχει η περίπτωση λόγω των διαφορετικών queries που γίνονται στη βάση, η εφαρμογή να καταλήξει με υπερβολικά περίπλοκα μοντέλα που κάνουν διαφορετικά πράγματα. Οπότε ο φόρτος εργασίας του μοντέλου κατά τις διαδικασίας αναζήτησης και ενημέρωσης να είναι συχνά ασύμμετρος με πολύ διαφορετικές απαιτήσεις απόδοσης.

Το CQRS pattern διαχωρίζει τις διαδικασίες αναζήτησης και εγγραφής δεδομένων στη βάση σε διαφορετικά μοντέλα, χρησιμοποιώντας εντολές για εγγραφή δεδομένων και queries για αναζήτηση δεδομένων. Οι εντολές πρέπει να βασίζονται σε tasks και όχι να επικεντρώνονται στα δεδομένα, επίσης μπορούν να τοποθετηθούν σε μία queue για ασύγχρονη επεξεργασία αντί να υποβάλλονται σε σύγχρονη επεξεργασία. Τα queries δεν τροποποιούν ποτέ τη βάση δεδομένων και κάθε query επιστρέφει ένα DTO (Data Transfer Object). Η ύπαρξη ξεχωριστών μοντέλων queries και ενημέρωσης απλοποιεί τον σχεδιασμό και την εφαρμογή. Ωστόσο, ένα μειονέκτημα είναι ότι ο κώδικας του CQRS pattern δεν μπορεί να δημιουργηθεί αυτόματα από ένα database schema χρησιμοποιώντας μηχανισμούς και εργαλεία όπως O/RM.

Για μεγαλύτερη απομόνωση, μπορείτε να διαχωρίσετε φυσικά τα δεδομένα αναζήτησης από τα δεδομένα εγγραφής στη βάση. Σε αυτήν την περίπτωση, η βάση δεδομένων ανάγνωσης μπορεί να χρησιμοποιήσει το δικό της σχήμα δεδομένων που είναι βελτιστοποιημένο για queries. Για παράδειγμα, μπορεί να αποθηκεύσει μια υλοποιημένη προβολή των δεδομένων, προκειμένου να αποφευχθούν σύνθετες συνδέσεις ή σύνθετες αντιστοιχίσεις O/RM. Μπορεί ακόμη και να χρησιμοποιήσει διαφορετικό τύπο αποθήκευσης δεδομένων. Για παράδειγμα, η βάση δεδομένων εγγραφής μπορεί να είναι σχεσιακή, ενώ η βάση ανάγνωσης μπορεί να είναι μια document database.

Εάν χρησιμοποιούνται ξεχωριστές βάσεις δεδομένων ανάγνωσης και εγγραφής, πρέπει να διατηρούνται σε συγχρονισμό. Συνήθως αυτό επιτυγχάνεται με το μοντέλο εγγραφής να δημοσιεύει ένα event κάθε φορά που ενημερώνει τη βάση δεδομένων. Η ενημέρωση της βάσης δεδομένων και η δημοσίευση του event πρέπει να πραγματοποιούνται σε μία μόνο συναλλαγή.

Diagram

Description automatically generated

Figure : CQRS architecture

Το read model μπορεί να είναι ένα αντίγραφο του write model μόνο για ανάγνωση ή τα read and write models μπορεί να έχουν διαφορετική δομή εντελώς. Η χρήση πολλαπλών αντιγράφων μόνο για το read μπορεί να αυξήσει την απόδοση του query, ειδικά σε κατανεμημένα σενάρια όπου τα αντίγραφα μόνο για ανάγνωση βρίσκονται κοντά στα instances της εφαρμογής.

Ο διαχωρισμός των read και write models επιτρέπει επίσης στο καθένα να κλιμακωθεί κατάλληλα για να ταιριάζει με το φορτίο. Για παράδειγμα, τα read models αντιμετωπίζουν συνήθως πολύ μεγαλύτερο φορτίο από τα write models.

Ορισμένες εφαρμογές του CQRS χρησιμοποιούν το Event sourcing pattern. Με αυτό το design pattern, η κατάσταση εφαρμογής αποθηκεύεται ως μια ακολουθία συμβάντων. Κάθε συμβάν αντιπροσωπεύει ένα σύνολο αλλαγών στα δεδομένα. Η τρέχουσα κατάσταση κατασκευάζεται με επανάληψη των γεγονότων. Σε ένα πλαίσιο CQRS, ένα πλεονέκτημα του Event sourcing είναι ότι τα ίδια συμβάντα μπορούν να χρησιμοποιηθούν για την ειδοποίηση άλλων στοιχείων - ιδιαίτερα, για την ειδοποίηση του read model. Το read model χρησιμοποιεί τα συμβάντα για να δημιουργήσει ένα στιγμιότυπο της τρέχουσας κατάστασης, το οποίο είναι πιο αποτελεσματικό για queries. Ωστόσο, το Event Sourcing προσθέτει πολυπλοκότητα στο σχεδιασμό.

Text

Description automatically generated with medium confidence

Figure 2: CQRS pattern with separated read and write stores

* Τα οφέλη του CQRS περιλαμβάνουν:
* Ανεξάρτητη κλιμάκωση. Το CQRS επιτρέπει την ανεξάρτητη κλιμάκωση του φόρτου εργασίας ανάγνωσης και εγγραφής.
* Βελτιστοποιημένα schema δεδομένων. Η πλευρά ανάγνωσης μπορεί να χρησιμοποιήσει ένα schema που είναι βελτιστοποιημένο για queries, ενώ η πλευρά εγγραφής χρησιμοποιεί ένα schema που έχει βελτιστοποιηθεί για ενημερώσεις.
* Ασφάλεια. Είναι ευκολότερο να διασφαλίσετε ότι μόνο οι σωστές οντότητες τομέα εκτελούν εγγραφή στα δεδομένα.
* Διαχωρισμός ανησυχιών. Ο διαχωρισμός των πλευρών ανάγνωσης και εγγραφής μπορεί να οδηγήσει σε μοντέλα που είναι πιο διατηρήσιμα και ευέλικτα. Το μεγαλύτερο μέρος της πολύπλοκης επιχειρησιακής λογικής μπαίνει στο μοντέλο εγγραφής. Το μοντέλο ανάγνωσης μπορεί να είναι σχετικά απλό.
* Απλούστερα queries. Αποθηκεύοντας μια υλοποιημένη προβολή στη βάση δεδομένων ανάγνωσης, η εφαρμογή μπορεί να αποφύγει σύνθετες συνδέσεις κατά την ερώτηση.

## Event Sourcing Pattern

Οι περισσότερες εφαρμογές λειτουργούν με δεδομένα και η τυπική προσέγγιση είναι η εφαρμογή να διατηρεί την τρέχουσα κατάσταση των δεδομένων ενημερώνοντάς τα καθώς οι χρήστες εργάζονται μαζί τους. Για παράδειγμα, στο παραδοσιακό μοντέλο create, read, update, delete (CRUD) μια τυπική διαδικασία δεδομένων είναι η ανάγνωση δεδομένων από το store, η πραγματοποίηση ορισμένων τροποποιήσεών του και η ενημέρωση της τρέχουσας κατάστασης των δεδομένων με τις νέες τιμές - συχνά χρησιμοποιώντας συναλλαγές που κλειδώνουν τα δεδομένα.

Η προσέγγιση CRUD έχει ορισμένους περιορισμούς:

Τα συστήματα CRUD εκτελούν λειτουργίες ενημέρωσης απευθείας κατά ενός datastore, το οποίο μπορεί να επιβραδύνει την απόδοση και την απόκριση και να περιορίσει την επεκτασιμότητα, λόγω της γενικής επιβάρυνσης επεξεργασίας που απαιτεί.

Σε έναν τομέα συνεργασίας με πολλούς ταυτόχρονους χρήστες, οι συγκρούσεις ενημέρωσης δεδομένων είναι πιο πιθανές επειδή οι λειτουργίες ενημέρωσης πραγματοποιούνται σε ένα μόνο στοιχείο δεδομένων.

Αν δεν υπάρχει ένας πρόσθετος μηχανισμός ελέγχου που καταγράφει τις λεπτομέρειες κάθε λειτουργίας σε ξεχωριστό αρχείο καταγραφής, το ιστορικό χάνεται.

Το Event coursing pattern καθορίζει μια προσέγγιση για τον χειρισμό λειτουργιών σε δεδομένα που καθοδηγείται από μια ακολουθία συμβάντων, καθένα από τα οποία καταγράφεται σε ένα store. Ο κώδικας της εφαρμογής αποστέλλει μια σειρά από events που περιγράφουν επιτακτικά κάθε ενέργεια που έχει συμβεί στα δεδομένα στο χώρο αποθήκευσης events. Κάθε event αντιπροσωπεύει ένα σύνολο αλλαγών στα δεδομένα.

Τα events διατηρούνται σε ένα event store που λειτουργεί ως σύστημα καταγραφής (ή έγκυρη πηγή δεδομένων) σχετικά με την τρέχουσα κατάσταση των δεδομένων. Το event store τυπικά δημοσιεύει αυτές τις εκδηλώσεις έτσι ώστε οι καταναλωτές να μπορούν να ειδοποιηθούν και να τις χειριστούν εάν χρειαστεί. Οι καταναλωτές θα μπορούσαν, για παράδειγμα, να ξεκινήσουν εργασίες που εφαρμόζουν τις λειτουργίες στα συμβάντα σε άλλα συστήματα ή να εκτελέσουν οποιαδήποτε άλλη σχετική ενέργεια που απαιτείται για την ολοκλήρωση της λειτουργίας. Παρατηρήστε ότι ο κώδικας εφαρμογής που δημιουργεί τα events αποσυνδέεται από τα συστήματα που έχουν εγγραφεί στα events.

Οι τυπικές χρήσεις των events που δημοσιεύονται από το event store είναι η διατήρηση υλοποιημένων απόψεων των οντοτήτων καθώς οι ενέργειες στην εφαρμογή τις αλλάζουν και η ενοποίηση με εξωτερικά συστήματα. Για παράδειγμα, ένα σύστημα μπορεί να διατηρήσει μια υλοποιημένη προβολή όλων των παραγγελιών πελατών που χρησιμοποιείται για τη συμπλήρωση τμημάτων του UI. Καθώς η εφαρμογή προσθέτει νέες παραγγελίες, προσθέτει ή αφαιρεί στοιχεία από την παραγγελία και προσθέτει πληροφορίες αποστολής, τα events που περιγράφουν αυτές τις αλλαγές μπορούν να αντιμετωπιστούν και να χρησιμοποιηθούν για την ενημέρωση της υλοποιημένης προβολής.

Επιπλέον, σε οποιοδήποτε σημείο είναι δυνατό για εφαρμογές να διαβάσουν το ιστορικό των γεγονότων και να το χρησιμοποιήσουν για να υλοποιήσουν την τρέχουσα κατάσταση μιας οντότητας αναπαράγοντας και καταναλώνοντας όλα τα events που σχετίζονται με αυτήν την οντότητα. Αυτό μπορεί να συμβεί κατά την υλοποίηση ενός αντικειμένου τομέα κατά τον χειρισμό ενός αιτήματος ή μέσω μιας προγραμματισμένης εργασίας, έτσι ώστε η κατάσταση της οντότητας να μπορεί να αποθηκευτεί ως υλοποιημένη προβολή για να υποστηρίξει το επίπεδο παρουσίασης.

Το σχήμα δείχνει μια επισκόπηση του μοτίβου, συμπεριλαμβανομένων ορισμένων επιλογών για τη χρήση της ροής συμβάντων, όπως δημιουργία υλοποιημένης προβολής, ενσωμάτωση συμβάντων με εξωτερικές εφαρμογές και συστήματα και επανάληψη συμβάντων για τη δημιουργία προβολών της τρέχουσας κατάστασης συγκεκριμένων οντοτήτων

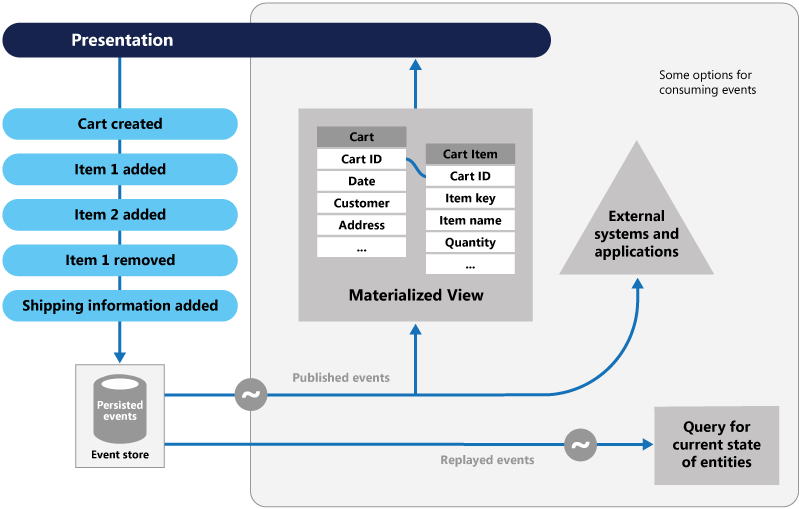


Figure 3: Event Sourcing pattern overview

Το Event Sourcing pattern παρέχει τα ακόλουθα πλεονεκτήματα:

* Τα events είναι αμετάβλητα. Η διεπαφή χρήστη, η ροή εργασίας ή η διαδικασία που ξεκίνησε ένα event μπορεί να συνεχιστεί και οι εργασίες που χειρίζονται τα events μπορούν να εκτελεστούν στο παρασκήνιο. Αυτό, σε συνδυασμό με το γεγονός ότι δεν υπάρχει διαφωνία κατά την επεξεργασία των συναλλαγών, μπορεί να βελτιώσει σημαντικά την απόδοση και την επεκτασιμότητα για εφαρμογές, ειδικά για το επίπεδο παρουσίασης ή τη διεπαφή χρήστη.
* Τα events είναι απλά αντικείμενα που περιγράφουν κάποια ενέργεια που συνέβη, μαζί με τυχόν συναφή δεδομένα που απαιτούνται για να περιγράψουν την ενέργεια που αντιπροσωπεύει το event. Τα events δεν ενημερώνουν απευθείας μια βάση δεδομένων. Απλώς καταγράφονται για χειρισμό την κατάλληλη στιγμή. Αυτό μπορεί να απλοποιήσει την εφαρμογή και τη διαχείριση.
* Τα events έχουν συνήθως νόημα για έναν domain expert, ενώ η αναντιστοιχία αντίστασης και αντικειμενικής σχέσης μπορεί να καταστήσει τους πολύπλοκους πίνακες βάσης δεδομένων δυσνόητους. Οι πίνακες είναι τεχνητές κατασκευές που αντιπροσωπεύουν την τρέχουσα κατάσταση του συστήματος και όχι τα γεγονότα που συνέβησαν.
* To event sourcing μπορεί να βοηθήσει στην αποφυγή σύγκρουσης ταυτόχρονων ενημερώσεων, επειδή αποφεύγει την απαίτηση άμεσης ενημέρωσης αντικειμένων στο χώρο αποθήκευσης δεδομένων. Ωστόσο, το μοντέλο πρέπει να έχει σχεδιαστεί για να προστατεύεται από αιτήματα που ενδέχεται να οδηγήσουν σε ασυνεπή κατάσταση.
* Η αποθήκευση events που προστίθεται μόνο στο παράρτημα παρέχει ένα ίχνος ελέγχου που μπορεί να χρησιμοποιηθεί για την παρακολούθηση ενεργειών που πραγματοποιούνται κατά ενός data store, την αναγέννηση της τρέχουσας κατάστασης ως υλοποιημένων προβολών ή προβολών με αναπαραγωγή των συμβάντων ανά πάσα στιγμή και τη βοήθεια στον έλεγχο και τον εντοπισμό σφαλμάτων του συστήματος. Επιπλέον, η απαίτηση χρήσης αντισταθμιστικών συμβάντων για ακύρωση αλλαγών παρέχει ένα ιστορικό αλλαγών που αντιστράφηκαν, κάτι που δεν θα συνέβαινε εάν το μοντέλο απλώς αποθηκεύσει την τρέχουσα κατάσταση. Η λίστα events μπορεί επίσης να χρησιμοποιηθεί για την ανάλυση της απόδοσης της εφαρμογής και τον εντοπισμό τάσεων συμπεριφοράς των χρηστών ή για τη λήψη άλλων χρήσιμων επιχειρηματικών πληροφοριών.
* Το event store αυξάνει γεγονότα και οι εργασίες που εκτελούν λειτουργίες ως απάντηση σε αυτά τα events. Αυτή η αποσύνδεση των εργασιών από τις εκδηλώσεις παρέχει ευελιξία και δυνατότητα επέκτασης. Οι εργασίες γνωρίζουν για τον τύπο του event και τα δεδομένα του event, αλλά όχι για τη λειτουργία που ενεργοποίησε το event. Επιπλέον, πολλαπλές εργασίες μπορούν να χειριστούν κάθε event. Αυτό επιτρέπει την εύκολη ενσωμάτωση με άλλες υπηρεσίες και συστήματα που ακούν μόνο νέα events που δημιουργούνται από το event store. Ωστόσο, τα events που προέρχονται από events τείνουν να είναι πολύ χαμηλού επιπέδου και μπορεί να χρειαστεί να δημιουργηθούν συγκεκριμένα γεγονότα ενσωμάτωσης.

## Retry pattern

Μια εφαρμογή που επικοινωνεί με στοιχεία που εκτελούνται στο cloud πρέπει να είναι ευαίσθητη στα παροδικά σφάλματα που μπορεί να προκύψουν σε αυτό το περιβάλλον. Τα σφάλματα περιλαμβάνουν τη στιγμιαία απώλεια σύνδεσης δικτύου με components και services, την προσωρινή μη διαθεσιμότητα μιας υπηρεσίας ή timeouts που συμβαίνουν όταν μια υπηρεσία είναι απασχολημένη.

Αυτά τα σφάλματα είναι συνήθως self-correcting(διορθώνονται μόνα τους) και εάν η ενέργεια που προκάλεσε ένα σφάλμα επαναληφθεί μετά από μια κατάλληλη καθυστέρηση, είναι πιθανό να είναι επιτυχής. Για παράδειγμα, μια υπηρεσία βάσης δεδομένων που επεξεργάζεται μεγάλο αριθμό ταυτόχρονων αιτημάτων μπορεί να εφαρμόσει μια στρατηγική πεταλούδας που απορρίπτει προσωρινά τυχόν περαιτέρω αιτήματα μέχρι να μειωθεί ο φόρτος εργασίας της. Μια εφαρμογή που προσπαθεί να αποκτήσει πρόσβαση στη βάση δεδομένων μπορεί να αποτύχει να συνδεθεί, αλλά αν προσπαθήσει ξανά μετά από καθυστέρηση μπορεί να πετύχει.

Στο cloud, τα παροδικά σφάλματα δεν είναι ασυνήθιστα και μια εφαρμογή θα πρέπει να σχεδιαστεί για να τα χειρίζεται ομαλά και χωρίς αυτό να είναι ευκρινές στον χρήστη. Αυτό ελαχιστοποιεί τις επιπτώσεις που μπορεί να έχουν τα σφάλματα στις επιχειρηματικές εργασίες που εκτελεί η εφαρμογή.

Εάν μια εφαρμογή εντοπίσει μια αποτυχία όταν προσπαθεί να στείλει ένα αίτημα σε μια απομακρυσμένη υπηρεσία, μπορεί να χειριστεί την αποτυχία χρησιμοποιώντας τις ακόλουθες στρατηγικές:

* Cancel. Εάν το σφάλμα υποδεικνύει ότι η αποτυχία δεν είναι παροδική ή είναι απίθανο να είναι επιτυχής εάν επαναληφθεί, η εφαρμογή θα πρέπει να ακυρώσει τη λειτουργία και να αναφέρει μια εξαίρεση. Για παράδειγμα, ένα authentication failure που προκαλείται από την παροχή invalid credentials δεν είναι πιθανό να επιτύχει, όσες φορές και αν επιχειρηθεί.
* Retry. Εάν το συγκεκριμένο σφάλμα που αναφέρεται είναι ασυνήθιστο ή σπάνιο, μπορεί να προκλήθηκε από ασυνήθιστες συνθήκες, όπως ένα network package που καταστρέφεται κατά τη μετάδοσή του. Σε αυτήν την περίπτωση, η εφαρμογή θα μπορούσε να ξαναδοκιμάσει το αποτυχημένο αίτημα ξανά, επειδή η ίδια αποτυχία είναι απίθανο να επαναληφθεί και το αίτημα πιθανότατα θα είναι επιτυχές.
* Retry after delay. Εάν το σφάλμα προκαλείται από μια από τις πιο συνηθισμένες αστοχίες συνδεσιμότητας ή απασχόλησης, το δίκτυο ή η υπηρεσία ενδέχεται να χρειαστούν ένα σύντομο χρονικό διάστημα ενώ τα ζητήματα συνδεσιμότητας διορθώνονται ή η εκκρεμότητα εργασιών διαγράφεται. Η εφαρμογή θα πρέπει να περιμένει για έναν κατάλληλο χρόνο πριν επαναλάβει το αίτημα.

Για τις πιο κοινές παροδικές αποτυχίες, το διάστημα μεταξύ των επαναλαμβανόμενων προσπαθειών θα πρέπει να επιλεγεί ώστε να διαδίδονται όσο το δυνατόν πιο ομοιόμορφα τα αιτήματα από πολλές περιπτώσεις της εφαρμογής. Αυτό μειώνει την πιθανότητα υπερφόρτωσης μιας πολυάσχολης υπηρεσίας. Εάν πολλές περιπτώσεις μιας εφαρμογής κατακλύζουν συνεχώς μια υπηρεσία με αιτήματα επανάληψης, θα χρειαστεί περισσότερος χρόνος για την ανάκτηση της υπηρεσίας.

Εάν το αίτημα εξακολουθεί να αποτυγχάνει, η εφαρμογή μπορεί να περιμένει και να κάνει άλλη προσπάθεια. Εάν είναι απαραίτητο, αυτή η διαδικασία μπορεί να επαναληφθεί με αυξανόμενες καθυστερήσεις μεταξύ των προσπαθειών επανάληψης, έως ότου επιχειρηθεί κάποιος μέγιστος αριθμός αιτημάτων. Η καθυστέρηση μπορεί να αυξηθεί σταδιακά ή εκθετικά, ανάλογα με τον τύπο της βλάβης και την πιθανότητα να διορθωθεί κατά τη διάρκεια αυτής της περιόδου.

Το παρακάτω διάγραμμα απεικονίζει την επίκληση μιας λειτουργίας σε μια φιλοξενούμενη υπηρεσία χρησιμοποιώντας αυτό το μοτίβο. Εάν το αίτημα είναι ανεπιτυχές μετά από προκαθορισμένο αριθμό προσπαθειών, η εφαρμογή θα πρέπει να αντιμετωπίζει το σφάλμα ως εξαίρεση και να το χειρίζεται ανάλογα.

Diagram

Description automatically generated

Figure 4:Retry pattern

# Δομή του Project

To συγκεκριμένο project είναι μία εφαρμογή κονσόλας, αναπαριστά μία δανειστική βιβλιοθήκη και απαρτίζεται από τα τρία design patterns που έχουμε αναλύσει παραπάνω (CQRS pattern, Event Sourcing pattern και Retry Pattern). Το βασικό κομμάτι κώδικα για το κάθε design pattern είναι βασισμένο σε παράδειγμα του Pluralsight ενώ η ενοποίηση των τριών patterns έχει υλοποιηθεί ειδικά για την συγκεκριμένη εργασία.

Graphical user interface, application

Description automatically generated

Figure : Project structure

Όπως φαίνεται στην παραπάνω εικόνα η

συνολική υλοποίηση του project απαρτίζεται από τέσσερα μεμονωμένα μικρότερα projects. Τα τρία από αυτά έχουν την υλοποίηση του κάθε design pattern ξεχωριστά ενώ το MainApp εξασφαλίζει την ενοποίηση των άλλων τριών.

## Δομή του CQRS Pattern

Graphical user interface, text

Description automatically generated

Figure 6:CQRS Pattern structure

To CQRS pattern. Έχει αναλάβει τη διαδικασία του CRUD ενός βιβλίου από τον βιβλιοθηκάριο. Όπως φαίνεται στην εικόνα έξι, στο CQRS pattern έχουμε δημιουργήσει τρία packages. Τα Models, Datastores και Services.

Μέσα στα models περιέχονται δύο μοντέλα, το Book.cs το οποίο χρησιμοποιείται για διαβάζουμε δεδομένα από τη βάση δεδομένων και το BookDetails.cs που χρησιμοποιείται για την εγγραφή των δεδομένων στη βάση.

Text

Description automatically generated

Figure 7: Books.cs class

Text

Description automatically generated

Figure : BookDetails.cs class

Στο package Datastores περιέχονται επίσης δύο τάξεις.

Η CommandDatastore.cs περιέχει μία μέθοδο η οποία δημιουργεί μια λίστα από BookDetails και η QueryDatastore.cs περιέχει μία μέθοδο που δημιουργεί μία λίστα από Book.

Text

Description automatically generated

Figure 9: CommandDatastore.cs

Text

Description automatically generated

Figure : QueryDatastore.cs

Στο package Services υπάρχουν όλα τα services που είναι υπεύθυνα για την λειτουργεία της εφαρμογής(BookCommandService.cs, BookQueryServuce.cs, ConsistencyWorker.cs, StorageService.cs)

Όπως είδαμε παραπάνω, μια εφαρμογή που χρησιμοποιεί CQRS pattern περιλαμβάνει τη δημιουργία δύο μοντέλων και όχι ενός: το μοντέλο εγγραφής(right model) και το μοντέλο ανάγνωσης(read model). Το μοντέλο εγγραφής είναι απόλυτα consistent δηλαδή είναι εγγυημένο ότι τα δεδομένα είναι ενημερωμένα κατά το χειρισμό μιας εντολής. Ωστόσο, το μοντέλο ανάγνωσης εφαρμόζεται συχνά χρησιμοποιώντας το τελικό consistency. Αυτό επιτρέπει την επεκτασιμότητα, μπορεί να υπάρχουν πολλοί event handlers που ενημερώνουν ταυτόχρονα μεμονωμένα μοντέλα ανάγνωσης και υποστηρίζουν την ανεξάρτητη αναδημιουργία μοντέλων ανάγνωσης. Ως συνέπεια τα μοντέλα που διαβάζονται από την εφαρμογή να είναι τελικά consistent.

Στο project αυτό για το consistency της εφαρμογής χρησιμοποιείται η τάξη ConsistensyWorker.cs. Η τάξη αυτή χρησιμοποιεί την QueryDataStore.cs και τα δύο μοντέλα (Book.cs, BookDetails.cs) για να βοηθήσει στο consistency της εφαρμογής σε σχέση με βασικές λειτουργείες όπως η δημιουργία ενός νέου βιβλίου στο σύστημα, την ενημέρωση της ημερομηνίας της πνευματικής ιδιοκτησίας του βιβλίου αλλά και της διαγραφής ενός βιβλίου από τη βάση δεδομένων.

Text

Description automatically generated

Figure : ConsistencyWorker.cs

Η τάξη BookCommandService.cs χρησιμοποιεί την τάξη ConsistencyWorker.cs για να υλοποιήσει τις παραπάνω λειτουργίες.

Text

Description automatically generated

Figure 12: BookCommantService.cs

Ενώ η BookQueryService.cs δεν χρησιμοποιεί καθόλου την ConsistencyWorker.cs αφού χρησιμοποιείται για να διαβάζει δεδομένα από την βάση και έτσι δεν χρειάζεται consistency.

Έτσι χρησιμοποιεί μόνο το read model.

Text

Description automatically generated

Figure :BookQueryService.cs

To StorageService.cs είναι κατά κύριο λόγο ένα service που χρησιμοποιείται για την υλοποίηση του RetryPattern. Στη συγκεκριμένη εργασία το StorageService έχει αντιγραφεί από το RetryPattern project στο CQRSPattern με σκοπό την ενοποίηση αυτών των δύο. To StarageService.cs το οποίο ουσιαστικά περιέχει όλη την λογική του RetryPattern, χρησιμοποιεί το μοντέλο Book.cs για την υλοποίηση της μεθόδου storeBook. Η μέθοδος storeBook αποθηκεύει ένα βιβλίο στο storage.

Text

Description automatically generated

Figure 14: StorageService.cs StoreBook method

## Δομή του EventSourcing Pattern

Graphical user interface, text, application

Description automatically generated

Figure 15: Δομή του EventSourcingPattern

Το EventSourcing project έχει αναλάβει την λειτουργεία δανεισμού βιβλίων από την βιβλιοθήκη σε κάποιο πελάτη. Στο project αυτό έχουν δημιουργηθεί δύο packages.

To package Commands περιέχει τις τάξεις BookrentalCommand.cs και BookReturnCommand.cs

H ταξη BookRentalCommant.cs είναι ένα μοντέλο στο οποίο αποθηκεύεται η τιμή με την οποία ένας χρήστης δανείζεται ένα βιβλίο.

Text

Description automatically generated

Figure 16: BookRentalCommant.cs

Όταν κάποιος πελάτης επιστρέψει τα βιβλία που έχει δανειστεί από την βιβλιοθήκη μας νωρίτερα από το προβλεπόμενο χρονικό διάστημα, τότε έχει μία μικρή έκπτωση. Η τιμή της έκπτωσης αποθηκεύεται στο μοντέλο BookReturnCommand.cs

Text

Description automatically generated

Figure 17: BookReturnCommand.cs

Στο πακέτο EventStore υπάρχει το μοντέλο Command το οποίο αποθηκεύει το Id του κάθε Event που λαμβάνει χώρα κατά την εκτέλεση της εφαρμογής.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 18: Command.cs

Επίσης το μοντέλο περιλαμβάνει την τάξη EventStore.cs η οποία είναι και η βασική τάξη του EventSourcing Pattern. H τάξη αυτή έχει αναλάβει να καταχωρεί τα νέα events στο eventStore με τη μέθοδο AppendTo Stream και να υπολογίζει και να επιστρέφει το τελικό ποσό που πρέπει να πληρώσει ο χρήστης με τη μέθοδο GetBalance.

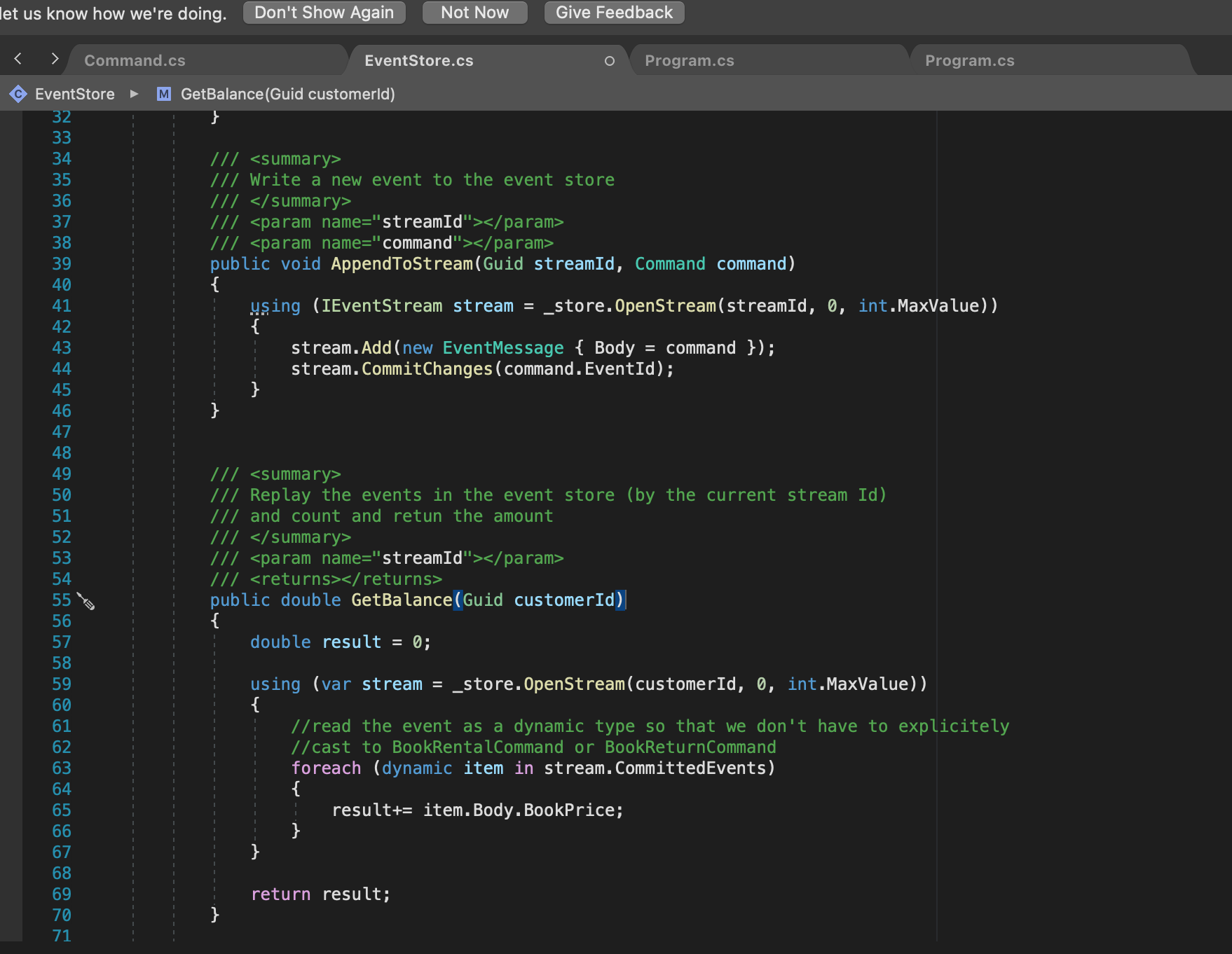


Figure 19: EventStore.cs

## Δομή του Retry Pattern

Graphical user interface, application

Description automatically generated

Figure 20: Retry Pattern Structure

Σε αντίθεση με τα προηγούμενα δύο design patterns παρατηρούμε ότι το retry pattern δεν περιέχει καθόλου καινούρια packages. Το retry pattern περιέχει μόνο μία τάξη, την StorageService.cs. Τη StorageService.cs τη χρησιμοποιήσαμε και παραπάνω στο CQRS Pattern για την ενοποίηση των δύο αυτών Patterns. Η βασική λειτουργία της StorageService.cs είναι αλληλεπίδραση της εφαρμογής με την remote database.

Με τη μέθοδο storageService γίνεται η σύνδεση στη βάση δεδομένων και καθορίζεται ο αριθμός των προσπαθειών επανασύνδεσης που θα κάνει το retry pattern σε περίπτωση αποτυχίας σύνδεσης με τη βάση δεδομένων. Με άλλα λόγια , τη στιγμή που η εφαρμογή τρέχει, για κάποιο λόγο υπάρχει περίπτωση να χαθεί η σύνδεση με τη βάση δεδομένων. Στην περίπτωση αυτή έχει προκαθοριστεί από τη μέθοδο storageService πόσες φορές θα ξαναγίνει προσπάθεια σύνδεσης και σε πόσο χρονικό διάστημα.

Η μέθοδος DisconectStorageService είναι υπεύθυνη για την αποσύνδεση της εφαρμογής από τη remote database όταν αυτό χρειάζεται.

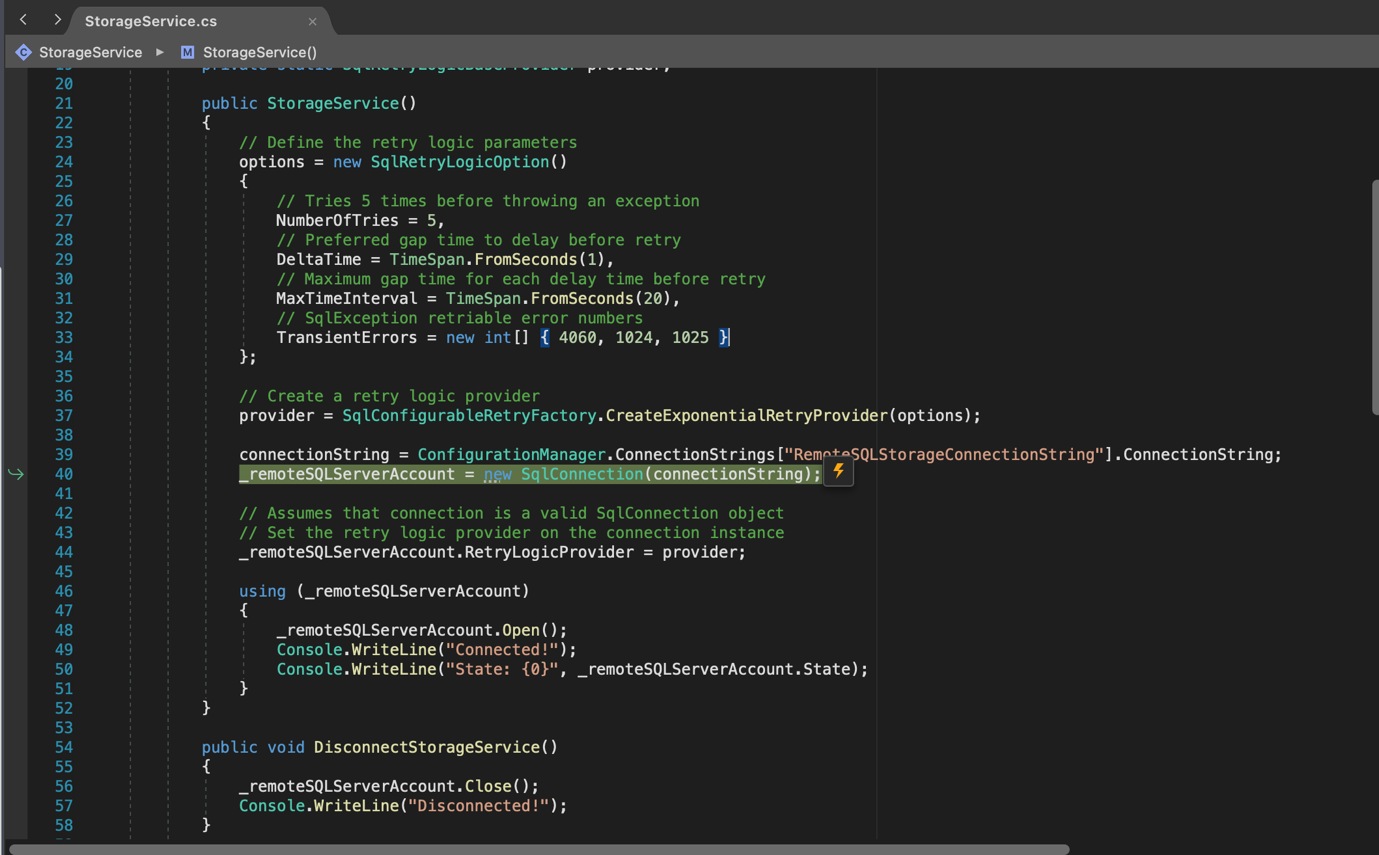


Figure 21: StorageService.cs. Methods: StorageService, DisconectStorageService

Επίσης στην StorageService υπάρχει και η μέθοδος StoreBook η οποία συνεργάζεται με το CQRS pattern για την αποθήκευση ενός νέου βιβλίου στη βάση δεδομένων. Η StoreBook χρησιμοποιεί το μοντέλο Book.cs που βρίσκεται στο CQRS pattern για την αποθήκευσή του στη βάση δεδομένων.

Text

Description automatically generated

Figure :StoreService.cs. Method: StoreBook

## Δομή του MainApp project και επεξήγηση της ενοποίησης των τριών design patterns.

Graphical user interface, application

Description automatically generated

Figure 23: MainApp project Structure

Το mainApp project αποτελείται μόνο από μία και μόνο τάξη, την τάξη Program.cs. Η τάξη αυτή είναι η κεντρική τάξη όλης της εφαρμογής και η τάξη στην οποία βλέπουμε ευκρινώς το πώς όλα τα design patterns συνδέονται μεταξύ τους. Η mainApp όπως βλέποθμε στην παρακάτω εικόνα, τρέχει διαδοχικά τρείς μεθόδους, την CQRSPattern(), την EventSourcingPattern() και την RetryPattern.



Figure 24: Main method of the project

Η μέθοδος CQRSPattern ως πρώτο της βήμα ανοίγει σύνδεση με τη βάση δεδομένων καλώντας την τάξη StorageService.Η τάξη StoarageService όπως είδαμε νωρίτερα ανήκει στο Retry Pattern και έτσι φαίνεται η ένωση μεταξύ RetryPattern και CQRSPattern.

Στη συνέχεια ζητάει από τον χρήστη να πατήσει ένα πλήκτρο έτσι ώστε να δημιουργηθούν δύο νέα βιβλία χρησιμοποιώντας τη μέθοδο CreateBook του CommandService.

H Τρίτη ενέργεια της μεθόδου είναι να ζητήσει από το χρήστη να πατήσει ένα δεύτερο πλήκτρο έτσι ώστε με τη βοήθεια του QueryService και της μεθόδου GetBook να μας εμφανίσει το βιβλίο που έχουμε ζητήσει. Στην περίπτωσή μας εμφανίζει το βιβλίο με id = 1.

Στη συνέχεια ζητείται από τον χρήστη άλλο ένα πάτημα πλήκτρου έτσι ώστε με τη βοήθεια πάλι του CommandService να αλλάξει την ημερομηνία έκδοσης του βιβλίου.

Ως Πέμπτη ενέργεια ζητάει ένα ακόμα πάτημα κουμπιού από τον χρήστη και με τη βοήθειά του QueryService και της μεθόδου GetBooksFromYear μας επιστρέφει όλα τα βιβλία από την χρονία που έχει ζητηθεί και μετά (στην περίπτωση μας από τη χρονιά 2017).

Τέλος, με τη βοήθεια του RetryPattern και της μεθόδου StoreBook αποθηκεύει στη βάση τα βιβλία. Εδώ βλέπουμε και μία δεύτερη χρήση του RetryPattern μέσα στο CQRSPattern.

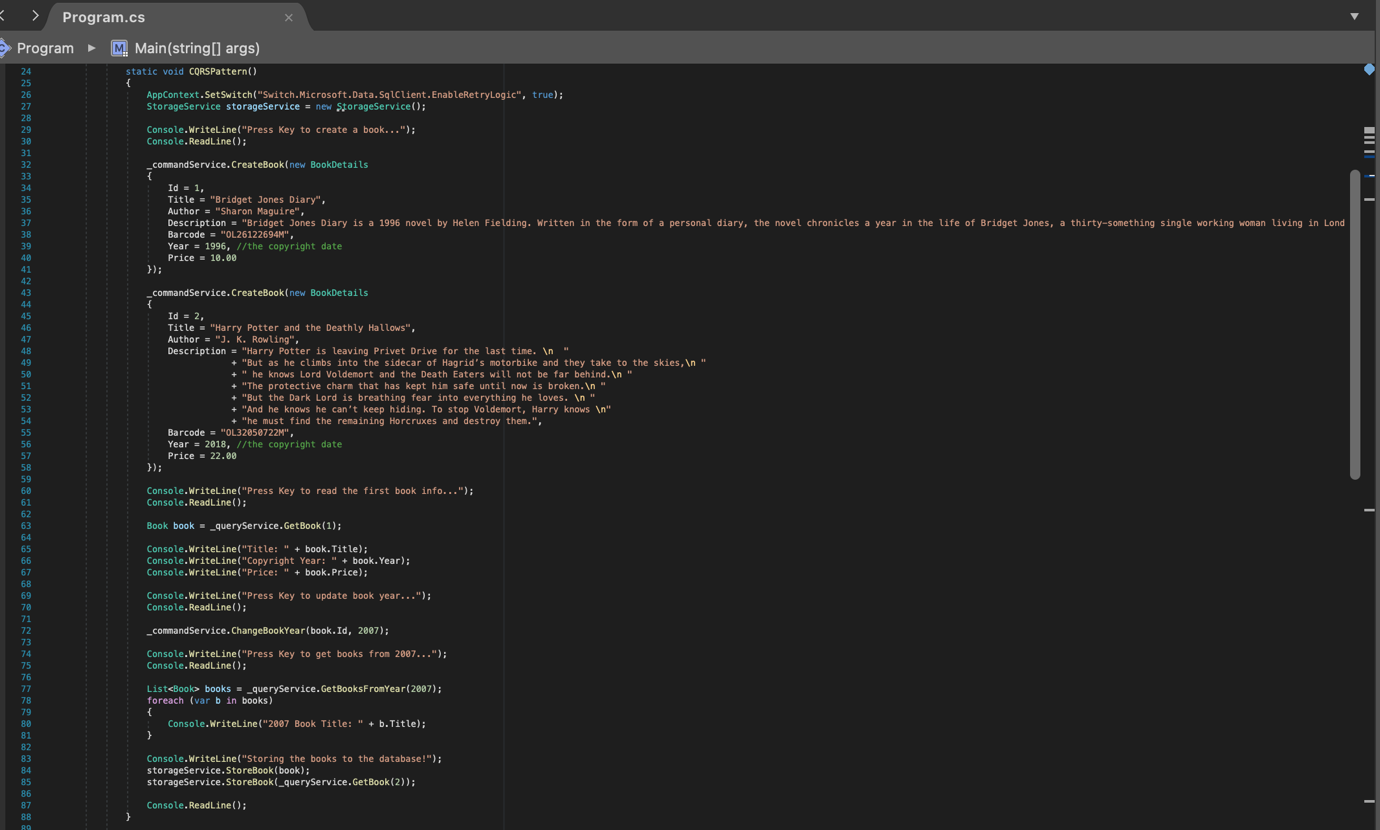


Figure 25: ΜainApp, Method: CQRSPattern

H μέθοδος EventSourcingPattern η οποία κατά βάση χρησιμοποιεί το ομόνοιμο pattern έχει την έννοια της δανειστικής βιβλιοθήκης. Βιβλία που υπάρχουν στο QueryDataStore μπορεί κάποιος να τα δανειστεί και να τα επιστρέψει.

Με την βοήθεια του BookQueryService από το CQRSPattern διαβάζει τα βιβλία. Πιο συγκεκριμένα μέσω της μεθόδου GetBook.

Για κάθε βιβλίο που διαβάζει δημιουργεί ένα event στο EventStore. Γίνεται δηλαδή όπως βλέπουμε στην παρακάτω εικόνα ενοικίαση 2 βιβλίων μέσω του BookRentalCommand και επιστροφή του ενός μέσω του BookReturnCommand.

Aυτά τα events γίνονται commit στο EventStore όπου τα δεδομένα αυτά καταλήγουν στον ίδιο remote MS SQL Server και στην ίδια βάση!

Και τέλος εμφανίζεται το balance του πελάτη που δανείστηκε και επέστρεψε τα βιβλία.

Text

Description automatically generated

Figure 26:MainApp, Method: EventSourcingPattern

Μόνο για λόγους πληρότητας χρηιμοποιείται και μια dummy μέθοδος του RetryPattern, η ReadWriteToRemoteStorage() για να φανεί και η ξεχωριστή χρήση του με αποθήκευση dummy δεδομένων στην βάση.Σε αυτό το σημείο δεν χρειάζεται να χρησιμοποιήσει το RetryPattern κάποιο από τα παραπάνω μιας και το CQRSPattern χρησιμοποιεί το RetryPattern και συγκεκριμένα την StoreBook για να αποθηκεύει τα βιβλία.



Figure : MainApp, Method: RetryPattern

Τέλος, όπως φαίνεται στην παραπάνω επεξήγηση, όλα τα design patterns αλληλοσυνδέονται. Συγκεκριμένα το CQRSPattern συνδέεται με το RetryPattern γιατι μέσω αυτού γίνεται η σύνδεση με την Remote MS SQL βάση δεδομένων και το EventSouncingPattern συνδέεται με το CQRSPattern αφού χρησιμοποιεί τα μοντέλα του για να μπορέσει να δημιουργήσει τη λειτουργία δανεισμού και επιστροφής των βιβλίων στη βιβλιοθήκη.

# Class Diagram

Graphical user interface, application, Word

Description automatically generated

# Πηγές

1. <https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>
2. <https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing>
3. <https://docs.microsoft.com/en-us/azure/architecture/patterns/retry>
4. <https://docs.microsoft.com/en-us/previous-versions/msp-n-p/dn589800(v=pandp.10)>
5. <file:///Users/sophia/Downloads/CloudDesignPatternsBook-PDF%20(3).pdf>
6. <https://10consulting.com/2017/10/06/dealing-with-eventual-consistency/>
7. <https://github.com/bmaluijb/CloudDesignPatterns>