ΠΡΟΤΥΠΑ ΑΝΑΠΤΥΞΗΣ ΛΟΓΙΣΜΙΚΟΥ

Π19204 – ΓΕΩΡΓΙΟΣ ΣΕΪΜΕΝΗΣ

p19204@unipi.gr

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

ΣΚΟΠΟΣ

Η εργασία είχε ως στόχο την μελέτη δύο (2) προτύπων ανάπτυξης λογισμικού από το βιβλίο της Microsoft Press «Cloud Design Patterns». Οι φοιτητές έπρεπε να επιλέξουν ένα από τα εξής ζεύγη προτύπων:

- Health Endpoint Monitoring Pattern, Competing Consumers Pattern
- Event Sourcing Pattern, Command and Query Responsibility Segregation (CQRS) Pattern
- Federated Identity Pattern, Retry Pattern

Το ζεύγος προτύπων που επελέγη ήταν το **Federated Identity Pattern και το Retry Pattern.**

ΥΛΟΠΟΙΗΣΗ

Χρησιμοποιώντας την Python 3.9 και το Visual Studio Code δημιουργήθηκαν παραδείγματα με τα δύο πρότυπα σε δύο ξεχωριστούς φακέλους. Ο κάθε φάκελος έχει το όνομα του προτύπου ανάπτυξης και μέσα σε αυτόν υπάρχει η υλοποίηση.

FEDERATED IDENTITY PATTERN

ΠΡΟΒΛΗΜΑΤΑ ΠΡΙΝ ΤΗ ΧΡΗΣΗ

Για να χρησιμοποιήσει κανείς το Federated Identity Pattern, σημαίνει ότι έχει κάποια εταιρεία, στην οποία **οι υπάλληλοι χρησιμοποιούν πολλούς διαφορετικούς λογαριασμούς για διαφορετικές εφαρμογές**. Δηλαδή:

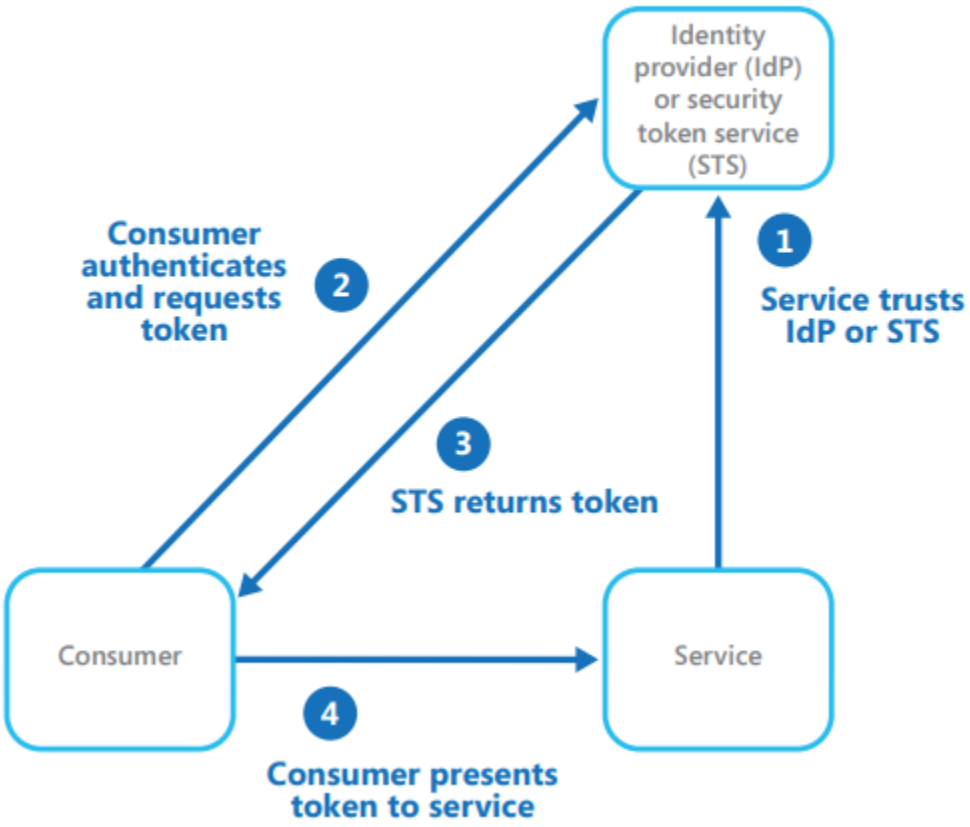
- Οι υπάλληλοι μιας εταιρείας πρέπει να απομνημονεύουν πολλούς διαφορετικούς κωδικούς για διαφορετικούς λογαριασμούς
- Με την αποχώρηση ενός υπαλλήλου θα πρέπει να διαγραφούν, άμεσα, όλοι οι λογαρισμοί του (κάτι που μοιάζει απίθανο, αν η εταιρεία χρησιμοποιεί πολλές εφαρμογές).
- Οι διαχειριστές της εταιρείας θα πρέπει να κρατάνε ασφαλή τα στοιχεία των λογαρισμών των υπαλλήλων. Συχνά, θα πρέπει να τους παρέχουν και βοήθεια, όπως π.χ. να τους δίνουν υπαινιγμούς σε περίπτωση που ξεχάσουν έναν κωδικό.

Όταν παρουσιάζονται τα παραπάνω προβλήματα σε μία εταιρεία, το Federated Identity Pattern έρχεται να τα λύσει, με μία αρκετά απλή προσέγγιση.

ΠΡΟΣΕΓΓΙΣΗ

Το Federated Identity Pattern παρουσιάζει ένα κεντρικό σύστημα ταυτοποίησης και κρυπτογράφισης στοιχείων για όλες τις εφαρμογές. **Οι χρήστες, χρειάζεται να κάνουν ταυτοποίηση στοιχείων μονάχα μία φορά, για να**

έχουν πρόσβαση σε όλες τις υπηρεσίες της εταιρείας. Όλοι οι κωδικοί των χρηστών κρυπτογραφούνται και κρατούνται ασφαλείς. Έτσι, οι υπάλληλοι κρατάνε την ομοσπονδιακή τους ταυτότητα (federated identity).



Εικόνα Α: Παράδειγμα ενός Class Diagram εφαρμογής που χρησιμοποιεί το Federated Identity Pattern

ΠΑΡΑΔΕΙΓΜΑΤΑ

Υποθέτουμε ότι έχουμε διάφορες κλάσεις, οι οποίες αναπαριστούν εφαρμογές. Τις ονομάζουμε Application A, Application B, Application C αντίστοιχα. Όλες αυτές οι εφαρμογές επιζητούν δημιουργία λογαριασμού και ταυτοποίηση στοιχείων. Αλλά, είναι διαφορετικές μεταξύ τους, δηλαδή δεν είναι ίδιοι οι

λογαριασμοί. Χρησιμοποιώντας το Federated Identity Pattern, φτιάχνουμε ένα κεντρικό σύστημα ταυτοποίησης, όπως αυτό φαίνεται παρακάτω:

```
from application_a import ApplicationA
from application_b import ApplicationB
from application_c import ApplicationC

class Auth:

    def __init__(self):
        self.registered_accounts = {}
        self.application_a = ApplicationA()
        self.application_b = ApplicationB()
        self.application_c = ApplicationC()

    def login(self, username, password) -> bool:
        if username not in self.registered_accounts.keys():
            print("Wrong credentials.")
            return False

        if self.registered_accounts[username] != password:
            print("Wrong credentials.")
            return False

        print(f"Welcome back, {username}!")
        return True

    def register(self, username, password) -> bool:
        if username in self.registered_accounts.keys():
            print("Username already exists.")
            return False

        print("Encrypting passwords...")
        self.registered_accounts.update( {username : password} )

        self.application_a.register(username, password)
        self.application_b.register(username, password)
        self.application_c.register(username, password)

        print(f"Welcome! Nice to have you, {username}. You have now access to all
applications!")
        return True

    def useful_function_a(self, username, password):
```

```
        self.application_a.login(username, password)
        self.application_a.useful_function()

    def useful_function_b(self, username, password):

        self.application_b.login(username, password)
        self.application_b.useful_function()

    def useful_function_c(self, username, password):

        self.application_c.login(username, password)
        self.application_c.useful_function()
```

Εικόνα Β: Παράδειγμα κώδικα από το Federated Identity Pattern.

Ο παραπάνω κώδικας είναι ενδεικτικό της ιδέα του Federated Identity Pattern. Η κλάση Auth περιέχει μέσα όλες τις διαθέσιμες εφαρμογές, που χρησιμοποιεί μια υποτιθέμενη εταιρεία. Μέσα σε αυτήν γίνεται κεντρική ταυτοποίηση και χρήση όλων των εφαρμογών, όταν ένας υπάλληλος χρειαστεί μία από αυτές.

RETRY PATTERN

ΧΡΗΣΗ

Το Retry Pattern είναι λίγο πιο απλό από το Federated Identity Pattern. **Σκοπός του Retry Pattern είναι να επαναλάβει μία διαδικασία, όταν αυτή αποτύχει.** Το πρότυπο αυτό, το βλέπουμε συνήθως να επαναλαμβάνει διαδικασίες που, συχνά, χρειάζονται δεύτερη ευκαιρία για να επιτύχουν. Π.χ., όταν μία ιστοσελίδα δεν ανταποκρίνεται, τότε ίσως χρειάζεται να επαναληφθεί η διαδικασία.

ΑΣΚΟΠΕΣ ΧΡΗΣΕΙΣ

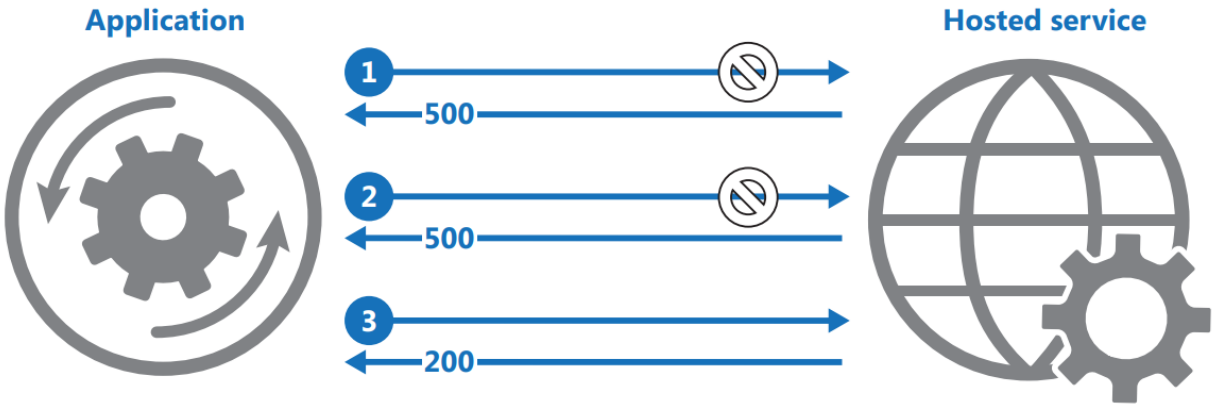
Κάποιες διαδικασίες, είναι καταδικασμένες να αποτύχουν, όσες φορές κι αν προσπαθεί το πρόγραμμα. Αν, για παράδειγμα, ένας χρήστης παραδίδει λάθος στοιχεία στο σύστημα, δεν υπάρχει λόγος να επαναληφθεί η διαδικασία. Στην δεύτερη φορά, ο χρήστης θα πρέπει να πληκτρολογήσει τον σωστό κωδικό. Οπότε, **δεν** έχει νόημα να χρησιμοποιήσουμε αυτό το πρότυπο εάν:

- Ξέρουμε ότι μία διαδικασία θα αποτυγχάνει για αρκετή ώρα.
- Το λάθος το έχει κάνει ο χρήστης.
- Ξέρουμε ότι ο εξυπηρετητής δεν δουλεύει ή έχει «πέσει».

Προφανώς, οι παραπάνω διαδικασίες, όπως είπαμε και πριν, είναι καταδικασμένες να αποτύχουν, όσες φορές κι αν επαναληφθούν.

ΕΦΑΡΜΟΓΗ

Για να τεθεί σε λειτουργία το πρότυπο αυτό, θα πρέπει να στηθεί για κάθε διαδικασία ένα σύστημα επανάληψης διαδικασιών (για λόγους επίδειξης, εδώ έχει γίνει με ξεχωριστή κλάση). Μέσα στις μεθόδους τις κλάσεις ορίζουμε έναν αριθμό προσπαθειών και ένα όριο. Αν η εφαρμογή εντοπίσει ότι έγινε κάποιο λάθος, θα επαναλάβει την διαδικασία. **Εάν ο αριθμός των προσπαθειών ξεπεράσει το όριο που έχουμε θέσει, τότε το πρόγραμμα θα σταματήσει να προσπαθεί να επαναλάβει την διαδικασία.** Βέβαια, όπως είπαμε και πριν, εξαρτάται κι από το πρόβλημα, το πόσες φορές θα επαναληφθεί μια διαδικασία. Δηλαδή, **στην περίπτωση που ο χρήστης έχει δώσει λάθος κωδικό πρόσβασης, το σύστημα δεν θα επαναλάβει την διαδικασία.**



Εικόνα Γ: Παράδειγμα διαγράμματος για το Retry Pattern

ΠΑΡΑΔΕΙΓΜΑ ΣΕ ΚΩΔΙΚΑ

```
    def retry_news_page(self):

        tries = 0
        while True:

            try:
                return self.server.get_central_news_page()
```

```
            except:
                tries += 1
                print(f"Failed attempt. (Tries: {tries})")

                if (tries > 5):
                    return "Too many failed attempts. Server is not responding."
```

Εικόνα Δ: Κώδικας για την ανάκτηση μίας σελίδας, με το Retry Pattern.

Στην παραπάνω διαδικασία, η εφαρμογή προσπαθεί να πάρει μια κεντρική σελίδα. Προσπαθεί το πολύ πέντε φορές, πριν ενημερώσει τον χρήστη ότι κάτι πάει λάθος.

```
    def retry_login(self, username, password):

        tries = 0
        while True:

            try:
                return self.server.login(username, password)
            except Error500:
                tries += 1
                print(f"Failed attempt. (Tries: {tries})")

                if (tries > 5):
                    return "Too many failed attempts. Server is not responding."
            except WrongCredentialsError:
                return "Wrong Username and/or Password."
```

Εικόνα Ε: Κώδικας για την ταυτοποίηση στοιχείων, με το Retry Pattern

Από την άλλη, στην παραπάνω εικόνα φαίνεται αυτό που προείπαμε για τα λανθασμένα στοιχεία. Αν ο χρήστης κάνει το λάθος, τότε η εφαρμογή δεν θα ξαναπροσπαθήσει. Αλλά, αν το λάθος το κάνει ο εξυπηρετητής, τότε η εφαρμογή θα δώσει και δεύτερη ευκαιρία στο σύστημα.

ΤΕΛΟΣ ΠΑΡΟΥΣΙΑΣΗΣ

Π19204 – ΓΕΩΡΓΙΟΣ ΣΕΪΜΕΝΗΣ – p19204@unipi.gr