ΠΛΗ-414 Αρχές Κατανεμημένων Συστημάτων

Project Εξαμήνου 2018

Σε ομάδες έως 4 ατόμων

Περιεχόμενα

Επισκόπηση

Θα αναπτύξετε ως κατανεμημένο σύστημα μια εφαρμογή διαχείρισης αρχείων εικόνων, το οποίο θα παρέχει μηχανισμό ταυτοποίησης χρηστών, δικαιώματα πρόσβασης σε συλλογές εικόνων, πολλαπλούς εξυπηρετητές για αποθήκευση και ανάκτηση αρχείων, δυνατότητα υποβολής σχολίων σε αρχεία, web περιβάλλον για θέαση εικόνων και API για προγραμματική χρήση του συστήματος.

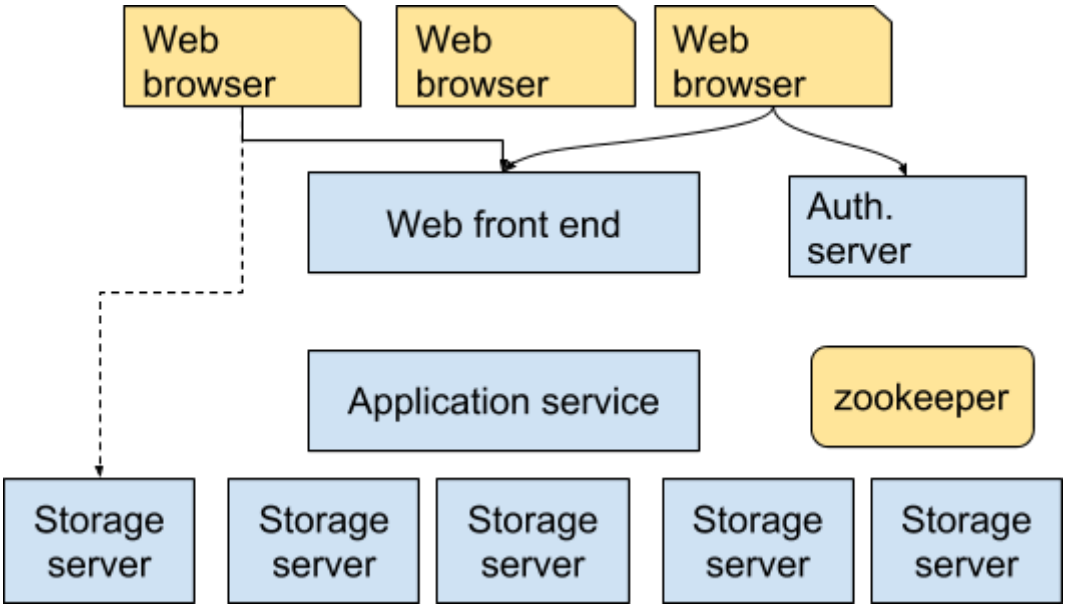
Περιγραφή απαιτήσεων (requirements)

1. Το σύστημα θα αναγνωρίζει χρήστες.
2. Κάθε χρήστης θα πρέπει να ταυτοποιηθεί με κάποιο authentication server που θα γνωρίζει το password του χρήστη.
3. Το application logic δε θα πρέπει να γνωρίζει, ούτε επιστρέπεται να ζητήσει το password κανενός χρήστη.
4. Για κάθε χρήστη U, το σύστημα θα αποθηκεύει ένα σύνολο από άλλους χρήστες, τους φίλους του U. Ο κάθε χρήστης θα μπορεί να διαλέξει ή να διαγράψει φίλους του μέσα από μια ιστοσελίδα. Η φιλία δεν είναι συμμετρική. Μπορεί ο Α να έχει φίλο τον Β αλλά ο Β να μην έχει φίλο τον Α.

5. Για κάθε χρήστη U, το σύστημα θα αποθηκεύει ένα σύνολο από galleries. Κάθε gallery G έχει όνομα (string) και περιλαμβάνει μια συλλογή εικόνων και ανήκει σε ένα μόνο ιδιοκτήτη (owner).
6. Ο ιδιοκτήτης ενός gallery είναι ο χρήστης που δημιούργησε το gallery.
7. Κάθε χρήστης U, καθώς και οι φίλοι του U, θα έχουν δικαίωμα να δούν τα περιεχόμενα (εικόνες) κάθε gallery που ανήκει στο U. Επίσης, θα έχουν δικαίωμα να προσθέσουν σχόλια στο gallery.
8. Ο χρήστης U που είναι ιδιοκτήτης του gallery G, μπορεί να εισάγει εικόνες, διαγράψει εικόνες, και να διαγράψει το ίδιο το G.
9. Μια εικόνα ανήκει σε ένα μόνο gallery.
10. Τα περιεχόμενα μιας εικόνας είναι ένα αρχείο που πρέπει να μπορεί να απεικονιστεί μέσα σε ένα web browser.
11. Τα περιεχόμενα μιας εικόνας αποθηκεύονται σε έναν αριθμό από Storage servers, από όπου θα πρέπει να τα προσπελαύνει ο browser του χρήστη, εφόσον έχει σχετικό δικαίωμα
12. Ένας χρήστης U μπορεί να δει μια εικόνα μαζί με τα σχόλιά της, καθώς και να προσθέσει σχόλια σε μια εικόνα, αν η εικόνα ανήκει σε κάποιο gallery που ο ιδιοκτήτης του έχει φίλο τον U.
13. Ένα σχόλιο είναι ένα κείμενο μέχρι 1024 χαρακτήρες.
14. Αν ένας χρήστης αφαιρέσει κάποιον άλλο χρήστη από τους φίλους του, τα σχόλια του πρώην φίλου παραμένουν στα αντικείμενα που ανήκουν.

Υλοποίηση

Κάθε ομάδα θα υλοποιήσει το σύνολο των παραπάνω συστημάτων. Το σύστημα ταυτοποίησης χρηστών θα πρέπει να υλοποιηθεί έτσι ώστε να μπορεί να χρησιμοποιηθεί από το σύστημα οποιασδήποτε ομάδας. Δηλαδή, η ταυτοποίηση των χρηστών θα μπορεί να γίνει είτε με το σύστημα που έχει αναπτύξει η ίδια η ομάδα, είτε με το σύστημα ταυτοποίησης των άλλων ομάδων. Για να το πετύχετε αυτό, το API της υπηρεσίας ταυτοποίησης θα πρέπει να είναι ακριβώς αυτό που περιγράφεται στην εκφώνηση.



Επιγραμματικά, τα τμήματα του συστήματος που πρέπει να αναπτύξετε είναι τα εξής:

- Application logic: υλοποιεί τη βάση δεδομένων του συστήματος και παρέχει ένα ReSTful API στο Web front end
- Storage server: Μια απλή υπηρεσία αποθήκευσης αρχείων. Κάθε στιγμιότυπο της υπηρεσίας δουλεύει ανεξάρτητα από τα υπόλοιπα. Ωστόσο, τα αρχεία εικόνων αποθηκεύονται σε περισσότερα από ένα server το καθένα, ώστε οι αναγνώσεις να είναι ταχύτερες.
- Web front end: εφαρμογή web για την εισαγωγή και επισκόπηση των δεδομένων
- Auth server: υπηρεσία ταυτοποίησης χρηστών

Επίσης, στη δομή της εφαρμογής σας θα πρέπει να χρησιμοποιήσετε το Apache ZooKeeper (https://zookeeper.apache.org/) για το configuration και την αρχικοποίηση του συστήματος.

Περιγραφή μη-λειτουργικών απαιτήσεων (non-functional requirements)

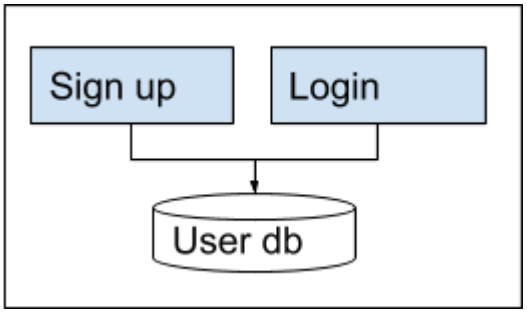
1. Το σύστημά σας θα πρέπει να εγκατασταθεί έτσι ώστε η κάθε υπηρεσία να βρίσκεται σε διαφορετικό υπολογιστή.
2. Αν ένα storage server τεθεί εκτός, θα πρέπει το σύστημα να μπορεί να συνεχίσει να λειτουργεί χωρίς κανένα πρόβλημα.
3. Προεραιτικά για bonus: αν κάποια web front end διεργασία πέσει, θα πρέπει το σύστημα να μπορεί να συνεχίσει να λειτουργεί χωρίς πρόβλημα.
4. Super bonus: πλήρης ανοχή σε σφάλματα: αν μέχρι μια διεργασία (οποιοδήποτε service) τεθεί εκτός, να μπορεί το σύστημα να συνεχίσει να λειτουργεί.

Περιγραφή των υποσυστημάτων της εφαρμογής

Authentication Service

Διαθέτει κατάλογο χρηστών και τους αναγνωρίζει με τη χρήση username/password. Παράγει ένα κρυπτογραφημένο μήνυμα το οποίο χρησιμοποιείται από τα άλλα μέρη του συστήματος για την αναγνώριση του χρήστη.

Authentication server



Interfaces:

- Web σελίδα SignUp για εισαγωγή username / password. Από αυτή τη σελίδα εγγράφεται ένας νέος χρήστης στην υπηρεσία.
- Web σελίδα Login με την οποία ταυτοποιείται ένας υπάρχων χρήστης.
- REST HTTP API

Τα interfaces είναι αυστηρά καθορισμένα ώστε να είναι δυνατή η διαλειτουργικότητα μεταξύ των συστημάτων που θα υλοποιήσουν οι ομάδες.

Storage Service

Ένα ή περισσότερα συστήματα στα οποία είναι αποθηκευμένα τα αρχεία. Παρέχει μηχανισμούς ανάκτησης και δημιουργίας αρχείων. Το κάθε αρχείο μπορεί να είναι αποθηκευμένο σε ένα μόνο storage service ή και σε περισσότερα (replication).

Συνεργάζεται με το Application Service για την υλοποίηση μηχανισμών εξουσιοδότησης του χρήστη και διαχείριση δεδομένων.

Το storage service πρέπει για κάθε κλήση, πριν την εξυπηρέτησή της, να εξασφαλίσει ότι υπάρχει σχετικό δικαίωμα πρόσβασης. Βλέπε ενότητα “Παρατηρήσεις για το Application Service” και αναφορά σε fingerprinting.

Interfaces:

- τουλάχιστον ένα REST HTTP API
- της επιλογής σας

Κάθε storage service κρατάει στατιστικά στοιχεία για τις προσβάσεις σε αυτόν. Τα στοιχεία αρκεί να κρατιούνται για τη διάρκεια της εκτέλεσης του storage service, και δε χρειάζεται να αποθηκεύονται μόνιμα. Θα πρέπει να κρατάει τουλάχιστον τα εξής:

- αριθμός read requests ανά λεπτό του χρόνου
- αριθμός write/delete requests ανά λεπτό του χρόνου

Application Service

Συντηρεί κατάλογο αρχείων και τα σημεία (storage service) στα οποία το κάθε ένα είναι αποθηκευμένο.

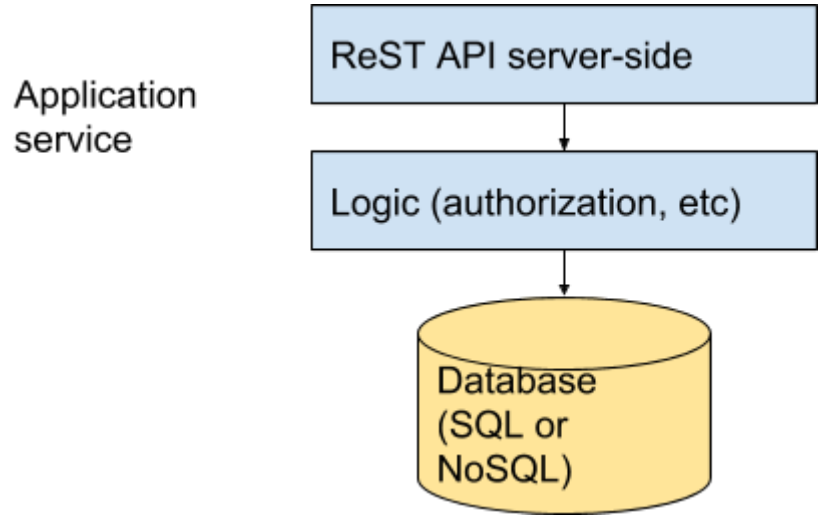
Ελέγχει τα δικαιώματα πρόσβασης του κάθε αρχείου.

Επιτρέπει την υποβολή σχολείων από τους χρήστες σε κάποιο αρχείο.

Αναθέτει αιτήματα για ανάκτηση ή δημιουργία αρχείων σε συγκεκριμένα storage service.

Η δημιουργία/εγγραφή/διαγραφή αρχείων γίνεται μόνο μέσω του Application Service. Κάθε αρχείο αποθηκεύεται σε περισσότερα του ενός storage services.

Για την ανάκτηση αρχείων, το application service επιλέγει ένα storage service στο οποίο είναι αποθηκευμένο το αρχείο, και το αναθέτει στον Client. Η ανάκτηση αρχείων καθ' αυτή γίνεται από τον Client με απ'ευθείας επικοινωνία με το storage service. Για αυτό το σκοπό, το application service στέλνει στον client σχετικά στοιχεία για την προσπέλαση στο αρχείο, με τρόπο που να μπορεί το storage service να επιβεβαιώσει το δικαίωμα πρόσβασης του client στο αρχείο. Βλέπε ενότητα "Παρατηρήσεις για το Application Service" και αναφορά σε fingerprinting.



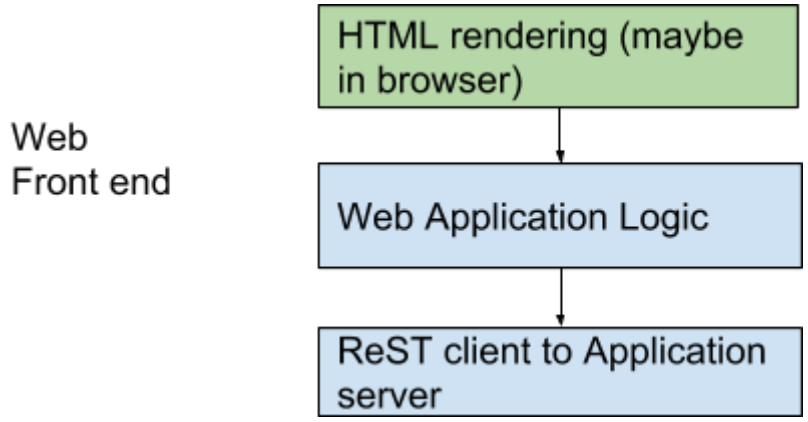
Interfaces:

- ένα REST HTTP API
- Επιπλέον: της επιλογής σας (πχ. για debugging)

Web Interface

Ένα Web Interface για browsing και δημιουργία galleries, ανέβασμα εικόνων, θέαση των εικόνων, καθορισμό φίλων, και υποβολή σχολείων για τα αρχεία και galleries.

Για την αναγνώριση των χρηστών, παραπέμπει στις σελίδες του ή των authentication services. Μετά την αναγνώριση του χρήστη, γίνεται ανακατεύθυνση πίσω στο web interface.



Interfaces: Web Σελίδες

Command Line εργαλείο (προεραιτικά)

Ένα command line εργαλείο που θα χρησιμοποιεί τα API των παραπάνω υπηρεσιών για την αναγνώριση του χρήστη και την υποβολή αρχείων προς αποθήκευση ή ανάκτηση αρχείων.

Πρέπει να δέχεται ως είσοδο το username password του χρήστη, και λίστα αρχείων προς υποβολή ή ανάκτηση.

Παραδοτέα

Για την εργασία σας θα πρέπει να παραδώσετε:

1. Τον πλήρη κώδικα όλων των στοιχείων του συστήματος που υλοποιήσατε (μαζί με Makefiles, Dockerfiles, etc)
2. Μια παρουσίαση σε powerpoint που θα περιγράφει τη ν υλοποίησή σας
    a. Εργαλεία και βιβλιοθήκες που χρησιμοποιήσατε
    b. Επιλογές που κάνατε (πχ., τεχνική υλοποίησης web pages)
    c. Εκτέλεση σε πάνω από 8 υπολογιστές (δείτε την υπηρεσία Okeanos στα Χρήσιμα) με μέτρηση απόδοσης (χρησιμοποιήστε το Apache Bench) που να δείχνει χρόνους εκτέλεσης σε concurrent requests
    d. Συμπεριφορά του συστήματος αν “πέσει” κάποιο service (e.g., storage server, zookeeper, application service)

3. Το παραπάνω powerpoint θα το παρουσιάσετε κατά την ημέρα παρουσίασης των projects, στο τέλος του εξαμήνου.

Παρατηρήσεις για το Application Service

Κάθε κλήση στην υπηρεσία, πριν εξυπηρετηθεί, ελέγχεται ως προς το δικαίωμα του πελάτη να πραγματοποιήσει τη συγκεκριμένη κλήση. Για αυτό το λόγο, κάθε κλήση πρέπει να συνοδεύεται από ένα κωδικοποιημένο μήνυμα με το οποίο διαπιστώνεται η ταυτότητα του καλούντος.

Το κωδικοποιημένο μήνυμα είναι το κρυπτογραφημένο Token που λήφθηκε κατά την αυθεντικοποίηση του χρήστη (βλέπε Authentication Service παρακάτω). Μέσω αυτού του Token το Directory Service γνωρίζει την ταυτότητα του χρήστη, οπότε και μπορεί να κρίνει αν είναι επιτρεπτή (βλέπε ενότητα με παρατηρήσεις για το Authentication Service και ενότητα για την κρυπτογράφηση μηνυμάτων)

Οι storage servers δεν ενδιαφέρονται για την ταυτότητα του καλούντος. Πρέπει να διασφαλίζουν όμως ότι είναι επιτρεπτή η εκάστοτε κλήση. Αυτό το επιτυγχάνουν με τη χρήση μηχανισμών fingerprting των παραμέτρων της κλήσης (βλέπε ενότητα για fingerprinting).

Ενδεικτικά endpoints κλήσεων που πρέπει να υποστηρίζει η υπηρεσία, με τα στοιχεία εισόδου και τις απαντήσεις:

GET /gallery/authToken
Output: list of galleries for user and all his friends

GET /gallery/authToken/<gallery-id>
Output: list of images and gallery comments for given gallery

GET /image/authToken/<image-id>
Output: image URL to some copy (at some Storage server), image comments.

POST /image
Input: image, authToken
Output: none

GET /friends/authToken/<user-id>
Output: friends of user-id

POST /friends/<user-id>/<friend-id>
Input: authToken
Output: none

DELETE /friends/authToken/<friend-id>
Output: none

Παρατηρήσεις για το Storage Service

Ένα ενδεικτικό API:

POST /createFile
Input: writeAccessToken, fileIdentifier, file
Output: -

GET /getFile/fileIdentifier/readAccessToken
Output: file

Παρατηρήσεις για το Web Interface

Μπορείτε να υλοποιήσετε το web interface όπως θέλετε. Μπορείτε να απευθυνθείτε στους διδάσκοντες και σε φίλους σας για συμβουλές.

Παρατηρήσεις για το Authentication Service

Το Authentication Service διατηρεί κατάλογο χρηστών και επιτρέπει την αναγνώρισή τους με χρήση ονόματος χρήστη και κωδικού πρόσβασης.
Μπορεί να χρησιμοποιηθεί από εξωτερικές υπηρεσίες για την αναγνώριση χρηστών. Για αυτό το σκοπό παρέχει 2 τρόπους πρόσβασης:

- REST API: η εξωτερική υπηρεσία καλεί το REST API (HTTP πρωτόκολλο) και δίνει το αναγνωριστικό της εξωτερικής υπηρεσίας, το όνομα χρήστη και τον κωδικό πρόσβασης. Επιστρέφεται μία δομή δεδομένων με το αναγνωριστικό του Authentication Service και ένα κρυπτογραφημένο μήνυμα με στοιχεία του χρήστη που αναγνωρίστηκε
- Web Interface με Callback: παρέχει ένα web interface. Μετά την επιτυχή αναγνώριση του χρήστη, γίνεται ανακατεύθυνση στην εξωτερική υπηρεσία. Κατά την ανακατεύθυνση, μεταδίδεται πάλι μία δομή δεδομένων με το αναγνωριστικό του Authentication Service και ένα κρυπτογραφημένο μήνυμα με στοιχεία του χρήστη που αναγνωρίστηκε. Η κλήση στο Web Interface γίνεται με παραμέτρους το αναγνωριστικό της εξωτερικής υπηρεσίας και τη διεύθυνση ανακατεύθυνσης.
  Το web interface παρέχει και σελίδα για εγγραφή νέων χρηστών.
  Το Web Interface του Authentication Service είναι διαφορετικό και ανεξάρτητο από το Web Interface του συστήματος που αναφέρθηκε παραπάνω!

Για να είναι δυνατή η χρήση ενός Authentication Service με μία εξωτερική υπηρεσία, πρέπει να δημιουργηθεί μία σχέση εμπιστοσύνης μεταξύ των. Το Authentication Service πρέπει να

δέχεται τη χρήση του από την εξωτερική υπηρεσία, και η εξωτερική υπηρεσία πρέπει να εμπιστεύεται τον Authentication Server.

Αυτό εξασφαλίζεται με τη χρήση κρυπτογραφημένων μηνυμάτων στα δεδομένα που επιστρέφονται για το χρήστη. Η κρυπτογράφηση γίνεται με κλειδί που γνωρίζει μόνο το Authentication Service και η συγκεκριμένη εξωτερική υπηρεσία.

Συγκεκριμένα:
Το Authentication Service γνωρίζει για κάθε εξωτερική υπηρεσία[1]

- το αναγνωριστικό της εξωτερικής υπηρεσίας,
- το κοινό κλειδί κρυπτογράφησης για τη συγκεκριμένη εξωτερική υπηρεσία,
- ένα όνομα για να το εμφανίζει στο χρήστη

Η εξωτερική υπηρεσία γνωρίζει

- το αναγνωριστικό του κάθε Authentication Service
- το κοινό κλειδί κρυπτογράφησης για το συγκεκριμένο Authentication Service,
- για το REST API:
    - τη διεύθυνση κλήσης του API
- για το web interface
    - τη διεύθυνση της σελίδας εισόδου

Για την εξασφάλιση της διαλειτουργικότητας του συστήματος μεταξύ των ομάδων, η δομή δεδομένων που επιστρέφεται από το Authentication Service είναι συγκεκριμένη, και ακολουθεί το πρότυπο JSON.

Για το REST API, τα δεδομένα επιστροφής είναι μήνυμα JSON με την εξής δομή

```
{
      error:"EMPTY|ERROR MESSAGE",
      data: "TOKEN"
}
```

Για το web interface, η κλήση στην callback διεύθυνση γίνεται με κλήση POST ή GET με παράμετρο με όνομα token και τιμή TOKEN

Το TOKEN είναι και αυτό ένα μήνυμα JSON με την εξής δομή

```
{
      error:"EMPTY|ERROR MESSAGE",
      issuer:"AUTHIDENTIFIER",
      crypted:"ENCRYPTEDMESSAGE"
}
```

Το `AUTHIDENTIFIER` χρειάζεται για να μπορεί η εξωτερική υπηρεσία να επιλέξει το σωστό κλειδί αποκρυπτογράφησης, που εξαρτάται από το Authentication Service που κρυπτογράφησε το μήνυμα.

[1] σε μία πραγματική υλοποίηση η διεύθυνση ανακατεύθυνσης από το web interface προς την εξωτερική υπηρεσία, θα ήταν συγκεκριμένη, ή θα περιοριζόταν από μία λίστα επιτρεπτών διευθύνσεων ανακατεύθυνσης. Οπότε ο Authentication Server θα είχε στοιχεία και για τις επιτρεπτές διευθύνσεις ανακατεύθυνσης

Το `ENCRYPTEDMESSAGE` είναι ένα κρυπτογραφημένο μήνυμα. Μετά την αποκρυπτογράφηση, έχουμε ένα μήνυμα σε μορφή JSON με την εξής δομή:

```
{
      "AUTHID":"AUTHIDENTIFIER",
      "SID":"SYSTEMIDENTIFIER",
      "userid":"sk@AUTHIDENTIFIER",
      "validtill":1519084800,
      "usermeta":{
            "name":"Stefanos Karasavvidis",
            "nick":"sk",
            "email":"sk@karasavvidis.gr"
      }
}
```

Το `SYSTEMIDENTIFIER` είναι το αναγνωριστικό της εξωτερικής υπηρεσίας.
Τα στοιχεία με έντονους χαρακτήρες είναι υποχρεωτικά να υπάρχουν. Στο usermeta μπορεί να υπάρχουν και επιπλέον στοιχεία/πληροφορίες για το χρήστη.

Για την κρυπτογράφηση των μηνυμάτων, θα χρησιμοποιηθεί συμμετρική κρυπτογράφηση AES-CBC-256. Χρησιμοποιεί κλειδί 32 byte μήκους. Θα σας δοθούν παραδείγματα κρυπτογράφησης/αποκρυπτρογράφησης για διάφορες γλώσσες προγραμματισμού (βλέπε ενότητα Κρυπτογράφηση / αποκρυπτογράφηση).

Για τη χρήση του Web Interface του Authentication Service, η εξωτερική υπηρεσία παρέχει στο χρήστη έναν σύνδεσμο προς το Authentication Service, ο οποίος έχει την εξής μορφή:
https://HOSTNAME/LOGINURL?callback=CALLBACKURL&system=SYSTEMIDENTIFIER
π.χ.
https://snf-811336.vm.okeanos.grnet.gr/skauth1/login.php?callback=https://snf-807664.vm.okeanos.grnet.gr/sksystem1/callback.php&system=SKSYSTEM

Χρήσιμα στοιχεία

Απόκτηση virtual machines από την υπηρεσία *Okeanos* για το project

Από την υπηρεσία https://cyclades.okeanos.grnet.gr/ του ΕΔΕΤ, κάθε φοιτητής μπορεί να δημιουργήσει έως 4 Virtual Machines με συνολικά έως 10 CPU και έως 4 IP διευθύνσεις ανά άτομο. Κάθε ομάδα θα πρέπει να χρησιμοποιήσει αυτούς τους πόρους για την ολοκλήρωση της εργασίας. **Προσοχή:** οι διαθέσιμοι πόροι στην υπηρεσία δεν επαρκούν για να χρησιμοποιήσουν όλοι οι εγγεγραμμένοι φοιτητές όλους τους προσωπικούς διαθέσιμους πόρους.

Για να αποκτήσετε πρόσβαση, κάντε login στην παραπάνω διεύθυνση, πηγαίνετε στο https://accounts.okeanos.grnet.gr/ui/projects και επιλέξτε Join. Ψάξτε για το project “tuc.gr" και αιτηθείτε τη συμμετοχή στο project. Η αίτησή σας θα εγκριθεί σύντομα, οπότε και θα μπορέσετε να δημιουργήσετε Virtual Machines.

Κρυπτογράφηση / αποκρυπτογράφηση

Για την κρυπτογράφηση / αποκρυπτογράφηση μηνυμάτων, θα χρησιμοποιήσετε τον αλγόριθμο AES-CBC-256. Χρησιμοποιεί κλειδί μήκους 32 byte.
Το κρυπτογραφημένο μήνυμα μεταδίδεται ως string που έχει 3 μέρη, χωρισμένα με άνω κάτω τελεία :
Το κάθε μέρος είναι κωδικοποιημένο ως base64 συμβολοσειρά.
Το πρώτο μέρος είναι το Initialization Vector (το χρειάζεται ο αλγόριθμος AES-CBC-256), το δεύτερο μέρος ένα HMAC (https://en.wikipedia.org/wiki/HMAC) και το τρίτο το κρυπτογραφημένο μήνυμα.
Ένα παράδειγμα είναι το εξής

```
XRZ8ZbL/lF0cBCFMxmPOag==:3W+ph5s53zMyIKISMUjha8YjosNa+U9CdmWaI2xTN
5g=:60if2nc8S/bKtGbP5+qaN8oIdg9mSerBqn5txrOD9TI=
```

Από τον παρακάτω σύνδεσμο μπορείτε να κατεβάσετε παραδείγματα κρυπτογράφησης και αποκρυπτογράφησης κειμένων για Java, Python, PHP και Javascript. Όλα τα παραδείγματα έχουν από μία συνάρτηση για κρυπτογράφηση και μία για αποκρυπτογράφηση. Η συγκεκριμένη υλοποίηση είναι για λήψη δεδομένων προς κρυπτογράφηση ή αποκρυπτογράφηση μέσω HTTP κλήσης.

https://filebox.isc.tuc.gr/index.php/s/q6GHiVsdBkMjmpH/download

Fingerprinting

Συχνά η μεταδιδόμενη πληροφορία δε χρειάζεται να είναι μυστική, αλλά πρέπει να εξασφαλίζεται ότι είναι έγκυρη ή επιτρεπτή ή δεν έχει αλλοιωθεί.

Στη συγκεκριμένη εφαρμογή, οι storage servers εξυπηρετούν μεταξύ άλλων αιτήσεις για ανάκτηση αρχείων, και εγγραφή σε αρχεία. Ενδεικτικές κλήσεις θα μπορούσαν να είναι οι εξής:
GET /file/file-id

POST /file
filedata=DATA
fileid=file-id

Και στις δύο περιπτώσεις, υποθέτουμε ότι τα δεδομένα καθ'αυτά δε μας πειράζει να υποκλαπούν[2]. Αλλά θέλουμε να εξασφαλίσουμε ότι η κλήση είναι επιτρεπτή. Ένας μηχανισμός για να το κάνουμε, είναι να παράγουμε ένα HMAC (https://en.wikipedia.org/wiki/HMAC) από τις παραμέτρους. Ένα περιορισμένου μήκους μήνυμα, με είσοδο τις παραμέτρους της κλήσης, και χρήση ενός μυστικού κλειδιού. Το HMAC μεταδίδεται μαζί με την κλήση.
Η άλλη πλευρά (ο storage server συγκεκριμένα), παράγει και ο ίδιος το HMAC από τις παραμέτρους της κλήσης, χρησιμοποιώντας το ίδιο κλειδί. Αν το HMAC που παρήγαγε ο storage server είναι το ίδιο με αυτό που του μεταδόθηκε με την κλήση, το μήνυμα είναι επιτρεπτό.
Στην πραγματικότητα, για την παραγωγή του HMAC, δε μας αρκούν οι παράμετροι για το file-id και το file-data. Το HMAC για το file-id θα είναι πάντα το ίδιο, ανεξάρτητα από το χρήστη που κάνει την πρόσβαση, και ανεξάρτητα με το πότε κάνει την πρόσβαση (π.χ. σήμερα είναι φίλος με ένα χρήστη, και μπορεί και βλέπει τις εικόνες του, αλλά αύριο δεν είναι πλέον φίλος). Πρέπει να περιορίσουμε περαιτέρω την πρόσβαση. Ένας μηχανισμός θα μπορούσε να είναι:

- τα HMAC αφορούν μόνο το συγκεκριμένο χρήστη που έκανε την πρόσβαση. Για την παραγωγή του HMAC χρησιμοποιούμε ως είσοδο και το αναγνωριστικό του χρήστη
- το HMAC είναι έγκυρο μόνο για περιορισμένο χρονικό διάστημα. Π.χ. 5 λεπτά. Οπότε για την παραγωγή του HMAC βάζουμε και το timestamp λήξης του χρόνου.

Χρησιμοποιώντας επιπλέον παραμέτρους για την παραγωγή του HMAC, πρέπει αυτές τις παραμέτρους να περάσουμε και σε plaintext μαζί με την κλήση, ώστε η άλλη πλευρά να μπορεί να αναπαράγει το HMAC και να το συγκρίνει.

Κώδικας παραγωγής HMAC υπάρχει στα παραδείγματα κρυπτογράφησης που αναφέρθηκε παραπάνω. Μέρος της κρυπτογράφησης είναι και η παραγωγή ενός HMAC για το κρυπτογραφημένο μήνυμα.

Αποθήκευση passwords χρηστών

Όταν σχεδιάζουμε ένα σύστημα αναγνώρισης χρηστών όπου αποθηκεύονται και κωδικοί χρηστών, δεν αποθηκεύουμε το password των χρηστών σε plaintext μορφή. Δε θέλουμε, κάποιος που αποκτάει πρόσβαση στη βάση δεδομένων που αποθηκεύει τα passwords, να αποκτάει πρόσβαση στα πραγματικά passwords των χρηστών.

Αυτό που κάνουμε είναι να παράγουμε ένα salted hash του password, και να αποθηκεύουμε αυτό στη βάση μας. Για να ελέγξουμε αν το password που έδωσε για την αναγνώριση ο χρήστης είναι σωστό, παράγουμε πάλι ένα hash και το συγκρίνουμε με αυτό που έχουμε αποθηκεύσει.

[2] Στην πραγματικότητα, στην κλήση POST για εγγραφή αρχείου, μας ενδιαφέρει να μην υποκλαπεί η εικόνα. Αλλά αυτό εξασφαλίζεται ποιο εύκολα π.χ. χρησιμοποιώντας κρυπτογράφηση σε επίπεδο HTTP πρωτοκόλλου (https σύνδεση).

Ο αλγόριθμος παραγωγής του hash, πέραν του ότι πρέπει να είναι “δύσκολος” να αντιστραφεί (να μπορεί δηλαδή κάποιος από το hash να βρει ένα password), πρέπει να είναι και αργός, ώστε η μαζική δοκιμή κωδικών να είναι υπολογιστικά ακριβή.

Παραδείγματα παραγωγής τέτοιων hash είναι τα παρακάτω

PHP

συναρτήσεις password_hash και password_verify

JAVA

Βλέπε βιβλιοθήκη jBCrypt http://www.mindrot.org/projects/jBCrypt/

Node.js / Javascript

Βλέπε https://www.abeautifulsite.net/hashing-passwords-with-nodejs-and-bcrypt

Python

Βλέπε ενότητα “Good Solution: bcrypt(password)”
http://dustwell.com/how-to-handle-passwords-bcrypt.html

HTTP Mime Type

Οι HTTP εξυπηρετητές, στις απαντήσεις τους προς μία κλήση, χρησιμοποιούν το Mime Type για να υποδηλώσουν το είδος των δεδομένων που στέλνουν στον πελάτη. Για το σκοπό αυτό χρησιμοποιούν το HTTP Header Content-Type: με τιμή το Mime Type. Κάποια Mime Type που πιθανώς να χρειαστείτε είναι τα εξής
`image/gif`: εικόνα τύπου GIF
`image/jpeg`: εικόνα τύπου JPEG
`image/png`: εικόνα τύπου PNG
`text/plain; charset=utf-8`: απλό κείμενο
`text/html; charset=utf-8`: κείμενο HTML
`application/json; charset=utf-8`: Κείμενο JSON

Οι HTTP Clients από τη μεριά τους, με βάση το Mime Type που τους δηλώνει ο εξυπηρετητής σε μίας απάντηση, χειρίζονται τα δεδομένα που λαμβάνουν.

Θα πρέπει λοιπόν σε κάθε απάντηση από τον Web Server σας να δηλώνετε το Mime Type, και ειδικά για δεδομένα κειμένου, και την κωδικοποίηση των χαρακτήρων που χρησιμοποιείτε.

Πέρασμα δεδομένων μέσω HTTP πρωτοκόλλου

Κλήσεις σε έναν HTTP Server γίνονται πάντα με χρήση ενός URI. Το HTTP πρωτόκολλο υποστηρίζει διάφορες μεθόδους κλήσεις (request methods). Οι πιο διαδεδομένες είναι η

HTTP GET (π.χ. κλικ σε ένα απλό link, εισαγωγή διεύθυνσης στον browser), και η HTTP POST μέθοδος (π.χ. ανέβασμα αρχείου μέσω φόρμας). Ένας πλήρης κατάλογος μεθόδων υπάρχει στη διεύθυνση https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Κατά την κλήση μπορούμε να υποβάλουμε δεδομένα σε μορφή key/value (request parameters). Ανάλογα με τη μέθοδο που χρησιμοποιούμε, η υποβολή γίνεται με διαφορετικό τρόπο.

Στην GET μέθοδο, η υποβολή γίνεται ως μέρος του URI. Συγκεκριμένα, βάζουμε ένα λατινικό ερωτηματικό στο τέλος, και ακολουθούν ζευγάρια key=value, χωρισμένα με &. Π..χ το παρακάτω URL έχει 2 παραμέτρους με 2 τιμές

```
http://xxx.yyy.zz/path/to/resource?paramname1=paramvalue1&paramname2=paramvalue2
```

Το μέρος του URI που αναφέρεται στις παραμέτρους, ονομάζεται query string

Στην POST μέθοδο, τα δεδομένα υποβάλλονται στο κυρίως μέρος της HTTP κλήσης. Π.χ.

```
POST /path/to/resource HTTP/1.1
Host: xxx.yyy.zz
paramname1=paramvalue1&paramname2=paramvalue2
```

Προσοχή1: Αν και το HTTP πρωτόκολλο δεν προβλέπει μέγιστο μήκος του URI σε GET μέθοδο, οι εφαρμογές πλοήγησης επιβάλουν μέγιστο μήκος, που είναι περίπου 2KB. Οπότε GET παράμετροι δεν ενδείκνυνται για μεγάλες ποσότητες δεδομένων

Προσοχή2: Οι επιτρεπτοί χαρακτήρες σε ένα URI είναι συγκεκριμένοι, και είναι κατά βάση λατινικοί χαρακτήρες και κάποιοι ειδικοί χαρακτήρες. Κάθε άλλος χαρακτήρας (αριθμητική τιμή του byte), πρέπει να μετατραπεί σε ειδική μορφή. Η αποκωδικοποίηση μίας ακολουθίας byte σε χαρακτήρες (π.χ. χαρακτήρες κωδικοποιημένοι με περισσότερα του ενός byte ανά χαρακτήρα), δεν καθορίζεται από το HTTP πρωτόκολλο. Ο σχετικός όρος URL encoding, και παρέχονται σε όλες τις γλώσσες σχετικές συναρτήσεις μετατροπής.

Προσοχή3: Για την αποθήκευση/μετάδοση χαρακτήρων με σειρά από bytes, αν και συνηθίζεται πλέον η κωδικοποίηση UTF-8, δεν μπορούμε να βασιστούμε σε αυτό και πρέπει σε κάθε μέρος της υποβολής και επεξεργασίας της κλήσης να έχουμε υπ'όψιν την κωδικοποίηση χαρακτήρων και να κάνουμε σχετικές μετατροπές.

Σημείωση: μία κλήση με μέθοδο POST, μπορεί στο URI να περιέχει και παραμέτρους τύπου GET

Ανάκτηση HTTP request attributes στην πλευρά του server

Κάθε γλώσσα, και πιθανώς framework, που χρησιμοποιούμε στην πλευρά του server για την εξυπηρέτησης HTTP κλήσεων, προσφέρει μεθόδους ανάκτησης των παραμέτρων που υποβλήθηκαν με την κλήση. Παρακάτω είναι κάποια παραδείγματα.

Παρακάτω δίνονται παραδείγματα για δεδομένα κειμένου. Για αρχεία, δείτε επόμενη ενότητα.

Παραδείγματα σε διάφορες γλώσσες για την κλήση του URL
`http://xxx.yyy.zz/path/to/resource?paramname1=paramvalue1&paramname2=paramvalue2`

Node.js / Javascript

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
var urlencodedParser = bodyParser.urlencoded({ extended: false });
app.get('/path/to/resource, urlencodedParser, function (req, res) {
       res.set({ 'content-type': text/plain; charset=utf-8' });
       res.send(req.body.input);
})
```

Αποτέλεσμα
`paramvalue1`

PHP

Προσφέρει τα arrays $_GET και $_POST, και το συνενωμένο $_REQUEST, που έχουν ως κλειδιά το όνομα της παραμέτρου και τιμή την τιμή της εκάστοτε παραμέτρου.

Δεν παρέχει τρόπο καθορισμού της κωδικοποίησης χαρακτήρων.

<php
```
header('Content-Type: plain/text; charset=utf-8');
echo $_GET['paramname1']
```

Αποτέλεσμα
`paramvalue1`

Java Servlet/JSP API

Η μεταβλητή `request` παρέχει μεθόδους πρόσβασης στις τιμές των παραμέτρων. Πρέπει να προηγηθεί κλήση του `request.setCharacterEncoding("UTF-8")` για τον καθορισμό της κωδικοποίησης των χαρακτήρων

```
<%@page language="java" contentType="plain/text;charset=UTF-8"
pageEncoding="UTF-8"%>
request.setCharacterEncoding("UTF-8");
```

```
System.out.println(request.getParameter("paramname1"));
```

Αποτέλεσμα
```
paramvalue1
```

Python

Με χρήση του cgi

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import cgi
print("Content-Type: plain/text;charset=utf-8")
print()
form = cgi.FieldStorage()
print form.getvalue("paramname1")
```

Αποτέλεσμα
```
paramvalue1
```

Ανάκτηση παραμέτρων που περνιούνται ως μέρος του resource identifier, χωρίς χρήση σύνταξης GET παραμέτρων

Για GET παραμέτρους που περνιούνται με σύνταξη ?paramname=paramvalue (query string), δόθηκαν παραπάνω παραδείγματα ανάκτησης των παραμέτρων. Όλα τα περιβάλλοντα ανάπτυξης web εφαρμογών (Java servlet, cgi, PHP κλπ), παρέχουν σχετικές συναρτήσεις.

Αν επιλέξετε όμως χρήση URL της μορφής /file/file-id το file-id δεν είναι μέρος του query string, αλλά μέρος του path. Οπότε δεν μπορείτε να χρησιμοποιήσετε τις συναρτήσεις που αφορούν query strings.

Αυτό που παρέχουν επίσης όλα τα περιβάλλοντα ανάπτυξης web εφαρμογών, είναι πρόσβαση στο request URL, το οποίο είναι ένα string.

Αρκεί να πάρετε αυτό το string, και να το χωρίσετε στα slashes (/) για να πάρετε τα μέρη που σας ενδιαφέρουν.

Σε μία παραγωγική εφαρμογή, θα χρησιμοποιούσατε πιθανώς κάποιο web framework για τη γλώσσα της επιλογής σας. Συχνά αυτά παρέχουν μηχανισμούς routing των αιτήσεων βάσει του μονοπατιού.

Χειρισμός δεδομένων JSON

Python

import json

```
obj = {
      'attr1':'val1',
      'attr2':'val2',
      'attr3':20,
};
string = json.JSONEncoder().encode(obj)
```

```
obj = json.JSONDecoder().decode(jsonString)
```

Java

Η Java δεν παρέχει εγγενή βιβλιοθήκη για χειρισμό JSON δεδομένων. Υπάρχουν όμως πάρα πολλές εξωτερικές βιβλιοθήκες που παρέχουν τέτοια δυνατότητα. Τα παρακάτω παραδείγματα χρησιμοποιούν τη βιβλιοθήκη https://github.com/stleary/JSON-java
Περισσότερα παραδείγματα θα βρείτε στο
https://www.codevoila.com/post/65/java-json-tutorial-and-example-json-java-orgjson

```
JSONObject obj = new JSONObject();
obj.put("attr1", "attr1");
obj.put("attr2", "attr2");
obj.put("attr3", 20);
String jsonString = result.toString();
```

Παράγει JSON String από στιγμιότυπο κλάσης org.json.JSONObject

```
String string = "{ \"name\":\"John\", \"age\":30, \"city\":\"New
York\"}";
JSONObject jsonObject = new JSONObject(string);
```

Από JSON string παράγει στιγμιότυπο κλάσης JSONObject.

PHP

Παρέχει την json_encode() και json_decode() συνάρτηση για χειρισμό JSON.

```
$array = [
      "attr1" => "val1",
      "attr2" => "val2",
      "attr3" => 20,
```

```
    ];
var $jsonString = json_encode($array);
```
Παράγει JSON String από PHP array.

```
$string = '{ "name":"John", "age":30, "city":"New York"}';
$array = json_decode(string, true);
```
Από JSON string παράγει PHP associative array.

Node.js / Javascript

Έχει εξαιρετική, εγγενή υποστήριξη JSON δεδομένων

```
var obj = {
    attr1:"val1",
    attr2:"val2",
    attr3:20,
};
```
Ορίζει μεταβλητή obj τύπου Object.

```
var myJSON = JSON.stringify(obj);
```
Μετατρέπει το obj σε JSON string.

```
var string = '{ "name":"John", "age":30, "city":"New York"}';
var obj = JSON.parse(string);
```
Από μεταβλητή τύπου string σε format JSON, δημιουργεί object.

Πραγματοποίηση κλήσης HTTP από Client Application

Θα χρειαστεί να πραγματοποιείτε κλήσεις προς HTTP εξυπηρετητές, περνώντας παραμέτρους, και να διαβάζετε και επεξεργάζεστε το αποτέλεσμα των κλήσεων.

Παρακάτω δίνονται παραδείγματα πραγματοποίησης τέτοιων κλήσεων για HTTP GET και POST μεθόδους με παραμέτρους τύπου string.

Σε μία πραγματική υλοποίηση θα χρειαστεί να χειριστείτε αρκετά πιθανά προβλήματα όπως προβλήματα στο δίκτυο, άρνηση πρόσβασης από τον εξυπηρετητή, προβλήματα στην επεξεργασία των απαντήσεων κλπ.

Παρακάτω δίνονται παραδείγματα για δεδομένα κειμένου. Για αρχεία, δείτε επόμενη ενότητα.

Java

POST

Χρησιμοποιούμε την `HttpURLConnection`

```
import java.io.*;
```

```
import java.net.*;
import java.util.*;

public class Post {
       public static void main(String[] args) {
         try {

               URL url = new URL("https://www.hostname.com/path/to/resource");
               HttpURLConnection conn = (HttpURLConnection) url.openConnection();
               conn.setDoOutput(true);
               conn.setRequestMethod("POST");
               conn.setRequestProperty( "Content-Type",
"application/x-www-form-urlencoded");
               conn.setRequestProperty( "charset", "utf-8");

               String input = "paramname1=paramvalue1&paramname2=20";
               byte[] postData       = input.getBytes( "UTF-8" );

               try( DataOutputStream wr = new DataOutputStream( conn.getOutputStream())) {
                  wr.write( postData );
               }

               Reader in = new BufferedReader(new InputStreamReader(conn.getInputStream(),
"UTF-8"));
               StringBuilder sb = new StringBuilder();
        for (int c; (c = in.read()) >= 0;)
            sb.append((char)c);
        String response = sb.toString();

               System.out.println("Output from Server .... \n");
               System.out.println(response);

               conn.disconnect();

         } catch (MalformedURLException e) {
               e.printStackTrace();
         } catch (IOException e) {
               e.printStackTrace();
        }
       }

}
```

GET

Χρησιμοποιούμε την `HttpURLConnection`

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;

public class Get {
       public static void main(String[] args) {
         try {
               URL url = new
URL("https://www.hostname.com/path/to/resource?paramname1=paramvalue1&paramname2=paramvalue
2");
               HttpURLConnection conn = (HttpURLConnection) url.openConnection();
```

```
                conn.setRequestMethod("GET");
                conn.setRequestProperty("Accept", "application/json");

                if (conn.getResponseCode() != 200) {
                        throw new RuntimeException("Failed : HTTP error code : "
                                        + conn.getResponseCode());
                }

                BufferedReader br = new BufferedReader(new InputStreamReader(
                        (conn.getInputStream())));

                String output;
                System.out.println("Output from Server .... \n");
                while ((output = br.readLine()) != null) {
                        System.out.println(output);
                }
                conn.disconnect();
          } catch (MalformedURLException e) {
                e.printStackTrace();
          } catch (IOException e) {
                e.printStackTrace();
          }
        }
}
```

PHP

Χρησιμοποιούμε την `file_get_contents()`.

POST

POST παράμετροι περνάμε μέσω της `$postdata`

```
<?php
$url = "https://www.hostname.com/path/to/resource";
$postdata = http_build_query(
        array(
                'paramname1' => "paramvalue1",
                'paramname2' => 20
        )
);
$opts = array('http' =>
        array(
                'method'  => 'POST',
                'header'  => 'Content-type: application/x-www-form-urlencoded',
                'content' => $postdata
        )
);
$context  = stream_context_create($opts);
// @ to suppress error messages
$response = @file_get_contents($url, false, $context);

if ($response === FALSE) {
        echo "Error retrieving response";
        // $http_response_header is automatically generated and filled with response
headers
        print_r($http_response_header);
}
else {
```

```
        // process response
        // e.g. parse json string
        echo $response;
}
```

GET

Πολύ πιο απλή κλήση σε σχέση με την POST. Οι παράμετροι, αν υπάρχουν, περνιούνται με τη διεύθυνση κλήσης

```
<?php
$url =
"https://www.hostname.com/path/to/resource?paramname1=paramvalue1&paramname2=paramvalue2";
$response = @file_get_contents($url);

if ($response === FALSE) {
        echo "Error retrieving response";
        // $http_response_header is automatically generated and filled with response
headers
        print_r($http_response_header);
}
else {
        // process response
        // e.g. parse json string
        echo $response;
}
```

Node.js / Javascript

Προσοχή: η javascript κάνει ασύγχρονες κλήσεις, και χειρίζεται τα αποτελέσματα με callback συναρτήσεις.

Απαιτεί `npm install request` (https://github.com/request/request)

POST

```
var request = require('request');
var url = "https://www.hostname.com/path/to/resource"

var formData = {
        'paramname1':'paramvalue1',
        'paramname2':'paramvalue2'
}
request.post(
        {
                url:url,
                formData: formData
        },
        function(error,response,data) {
                if (error != null) {
                        // handle error
                        console.log('handling error '+error);
                }
                else if(response && response.statusCode == 200) {
                        // process data
                        console.log('processing data');
```

```
                        console.log(data);
                }
                else {
                        // handle status code != 200
                        console.log('handling other status code');
                }
        }
);
```

GET

```
var request = require("request");
var url =
"https://www.hostname.com/path/to/resource?paramname1=paramvalue1&paramname2=paramvalue2";
var request = require('request');
request(url, function (error, response, data) {
        if (error != null) {
                // handle error
                console.log('handling error '+error);
        }
        else if(response && response.statusCode == 200) {
                // process data
                console.log('processing data');
        }
        else {
                // handle status code != 200
                console.log('handling other status code');
        }
});
```

Χειρισμός αρχείων μέσω HTTP POST μεθόδου

Για να μεταφέρει αρχεία ένας HTTP Client, και να τα χειριστεί ο HTTP server, ακολουθείται ένας ελαφρώς διαφορετικός τρόπος. Αν θέλουμε δηλαδή να στείλουμε ένα αρχείο σε έναν HTTP server, η κλήση της POST μεθόδου είναι λίγο διαφορετική.

Στην περίπτωση που χρησιμοποιείται HTML φόρμα για μεταφορά αρχείου, οπότε την κλήση την κάνει ο browser, υπάρχει διαφορά στο HTML στοιχεία της φόρμας. Μία απλή φόρμα, χωρίς αρχείο, θα οριζόταν κάπως έτσι

```
<form action="URL" method="POST" enctype="application/x-www-form-urlencoded">
        <input type="text" name="paramname1" value="">
        <input type="text" name="paramname1" value="">
        <input type="submit">
</form>
```

Στην περίπτωση που η φόρμα περιέχει και αρχείο, αλλάζει το `enctype`

```
<form action="URL" method="POST" enctype="multipart/form-data">
        <input type="text" name="paramname1" value="">
        <input type="file" name="someFile1" value="">
        <input type="submit">
</form>
```

Αν την κλήση την κάνουμε από δικό μας HTTP Client, πρέπει να λάβουμε αυτή τη διαφορά υπ' όψιν.

Για την εργασία, θα χρειαστεί να εκτελείτε POST κλήσεις για να μεταφέρετε α) το αρχείο, και β) δεδομένα που πιστοποιούν το δικαίωμα πρόσβασής σας

Java

Για Java, θα βρείτε σχετικό παράδειγμα κλήσης στη διεύθυνση
http://java-monitor.com/forum/showthread.php?t=4090
Σε αυτή τη διεύθυνση ορίζεται μία κλάση MultipartUtility, την οποία μπορείτε να τη χρησιμοποιήσετε ως εξής

```
public class Post {
	public static void main(String[] args) {
		final File uploadFile = new File("PATH_TO_FILE");
		try {
			URL url = new URL("https://www.hostname.com/path/to/resource");
			final MultipartUtility http = new MultipartUtility(url);
			http.addFormField("paramname1", "paramvalue1");
			http.addFilePart("someFile1", uploadFile);
			final byte[] bytes = http.finish();
			String resultJson = new String(bytes, StandardCharsets.UTF_8);
		} catch (IOException e) {
			e.printStackTrace();
		}

	}

}
```

Στο παραπάνω παράδειγμα, μεταφέρεται μία παράμετρος ως κείμενο (paramname1), και ένα αρχείο (someFile1).

Από την πλευρά του εξυπηρετητή, από το Servlet API 3 και άνω, υπάρχει υποστήριξη για χειρισμό αρχείων. Σχετικό παράδειγμα θα βρείτε στη διεύθυνση
http://www.codejava.net/java-ee/servlet/java-file-upload-example-with-servlet-30-api
Προσοχή: Στη συγκεκριμένη σελίδα, γίνεται χειρισμός αποκλειστικά αρχείων. Εάν στη δική σας κλήση έχετε τόσο αρχεία όσο και strings, πρέπει να κάνετε το διαχωρισμό μέσω των μεθόδων που προσφέρει η κλάση javax.servlet.http.Part και να τα χειριστείτε ανάλογα.

PHP

Για την κλήση χρησιμοποιούμε το curl με σχετική βιβλιοθήκη PHP

```
<?php
$url = 'https://www.hostname.com/path/to/resource';
$fields = [
	'someFile1' => new CurlFile(realpath('PATH_TO_FILE')),
```

```
        'paramname1' => 'paramvalue1'
];

$request = curl_init();
curl_setopt($request, CURLOPT_URL, $url);
curl_setopt($request, CURLOPT_HEADER, false);
curl_setopt($request, CURLOPT_POST, true);
curl_setopt(
                $request,
                CURLOPT_POSTFIELDS,
                $fields
    );
curl_setopt($request, CURLOPT_RETURNTRANSFER, true);
$result = curl_exec($request);
curl_close($request);
echo $result;
```

Από την πλευρά του εξυπηρετητή, στη μεταβλητή $_POST συνεχίζουμε να έχουμε τις παραμέτρους που δεν είναι αρχεία, ενώ στο μεταβλητή $_FILES έχουμε τα στοιχεία που αφορούν τα αρχεία. Η PHP αποθηκεύει το αρχείο σε ένα προσωρινό μονοπάτι `$_FILES["somefile1"]["tmp_name"]`, από το οποίο μπορούμε να το μεταφέρουμε στον τελικό προορισμό.

Node.js / Javascript

Παράδειγμα κλήσης POST με Javascript φαίνεται παρακάτω. Να σημειωθεί ότι ο κώδικας είναι όμοιος είτε τον εκτελείτε με Node.js είτε μέσα σε μία HTML σελίδα. Η διαφορά στην HTML σελίδα είναι ότι το αρχείο προέρχεται από φόρμα HTML, ενώ σε Node.js από το File System.

```
var fs = require('fs');
var request = require('request');
var formData = {
        paramname1: 'paramvalue1',
        someFile1: fs.createReadStream('PATH_TO_FILE')
};
request.post({url:'https://www.hostname.com/path/to/resource', formData: formData},
function(err, httpResponse, body) {
        if (err) {
                return console.error('upload failed:', err);
        }
        console.log('Upload successful!  Server responded with:', body);
});
```