Πανεπιστήμιο Αιγαίου

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

321-7951 Κατανεμημένα Συστήματα

Διδάσκων: Γκουμόπουλος Χρήστος

Πρώτη Ομαδική Εργαστηριακή Άσκηση

Υλοποίηση Rest Υπηρεσίας & Αξιοποίησή της

Εργαστηριακοί Συνεργάτες: Φακής Αλέξανδρος

Χαράλαμπος Αντωνιάδης

icsd10011

Θεοφάνης Μπινιάκου

icsd13126

Ευκαρπίδης Κωνσταντίνος

icsd15051

Σάμος, 11/6 , 2018

Ευκαρπίδης Κωνσταντίνος – icsd15051 , Χαράλαμπος Αντωνιάδης – icsd10011, Θεοφάνης Μπινιάκου – icsd13126

Περιεχόμενα

Ευκαρπίδης Κωνσταντίνος – icsd15051 , Χαράλαμπος Αντωνιάδης – icsd10011, Θεοφάνης Μπινιάκου – icsd13126

1 Εισαγωγή

Στην εργασία αυτή καλούμαστε να μελετήσουμε την υπηρεσία Rest για ένα μέσο κοινωνικής δικτύωσης. Θα υλοποιείτε με τις βιβλιοθήκες του Jersey και οι πληροφορίες που θα ανταλλάσσονται θα είναι στην μορφή JSON. Ακολουθεί το μοντέλο CRUD , αφού οι διαδικασίες που λαμβάνουν χώρα στην εφαρμογή είναι CREATE,READ,UPDATE &DELETE. Η εφαρμογή «InstaFace» περιλαμβάνει βασικές λειτουργίες των εφαρμογών αυτών. Παράλληλα πρέπει να δημιουργήσουμε και το documentation της εφαρμογής μας προκειμένου να υπάρχει δυνατότητα από άλλους προγραμματιστές να το χρησιμοποιήσουν για δική τους δουλειά. Τέλος το project ολοκληρώνεται συμπεριλαμβανομένου και της εφαρμογής του πελάτη ,ο οποίος θα συνδέεται στην υπηρεσία μας μέσα από ένα απλό γραφικό περιβάλλον (SWING).

Ευκαρπίδης Κωνσταντίνος – icsd15051 , Χαράλαμπος Αντωνιάδης – icsd10011, Θεοφάνης Μπινιάκου – icsd13126

2 Βάση Δεδομένων

Η υλοποίηση της βάσης είναι γραμμένη σε PL/SQL(Oracle).Διαλέξαμε αυτήν γιατί ήταν πάρα πολύ εύκολη η δημιουργία βάσης κα παραμετροποίησης της μέσω του NetBeans IDE. Για την σχεδίαση της βάσης επιλέξαμε να κάνουμε τέσσερις πίνακες. (PROFILE, POSTS, FRIENDS, USERS). Ο πίνακας Users περιέχει το id (PK) των χρηστών καθώς και τον κωδικό τους, ο πίνακας Posts το id (PK) του post, το id (FK) του profile που έγινε το post καθώς και το id (FK) του χρήστη που το έκανε, καθώς και το περιεχόμενο του. Ο πίνακας FRIENDS έχει το id της φιλίας καθώς και τα δύο id (FK) των χρηστών που αντιπροσωπεύουν την σχέση της φιλίας δύο χρηστών. Ο τέταρτος πίνακας είναι ο PROFILE περιέχει το id (PK) του profile , το id (FK) του χρήστη που αναφέρεται καθώς και λοιπά πεδία που αναφέρονται σε πληροφορίες του χρήστη. Πέρα από τους 4 πίνακες δημιουργήσαμε και 3 sequences για την αυτόματη αύξηση του P_ID,POST_ID & F_ID.

3 Δημιουργία Server

Για την δημιουργία του Server επιλέξαμε «**Maven Project**» και έπειτα «**Project from Archetype**». Στην συνέχεια προσθέσαμε τα απαραίτητα dependencies προκειμένου να είναι διαθέσιμες όλες οι βιβλιοθήκες που θα χρειαστούμε. Τα dependencies που προσθέσαμε είναι αυτά που παραθέτονται στο 11ο εργαστηριακό φυλλάδιο (Jersey). Ο Server περιέχει δύο πακέτα. Ένα είναι το default που δημιουργείτε κατά την δημιουργία του Project και ένα που φτιάξαμε εμείς προκειμένου να εισάγουμε τα Services μας εκεί. Δεν κάνουμε κάποια παράθεση στον κώδικα των κλάσεων Users,Profile,Post διότι περιέχουν μόνο απαραίτητα πεδία ,constructors, setters&getter (όπου χρειάζονται) και display μέθοδοι.

3.1 Services

Στο αρχείο user_service.java εμπεριέχεται ένα πολύ σημαντικό κομμάτι της εφαρμογής. Για την ακρίβεια περιέχει όλες τον κώδικα υλοποίησης των υπηρεσιών που προσφέρει η εφαρμογή μας. Οι λειτουργίες που ζητούνται και υλοποιούνται είναι :

- Δημιουργία λογαριασμού & σύνδεση στην υπηρεσία (**adduser** , **login**)
- Προσθήκη φίλου μέσω username (**addFriend**)
- Δημιουργία post στο προφίλ συνδεδεμένου χρήστη & φίλου (**addPost**)
- Ανάκτηση φίλων & προβολή πληροφοριών αυτών (**searchMyFriends** , **showMyProfile**)
- Ανάκτηση των 10 πιο πρόσφατων post στο προφίλ του χρήστη (**showPost**)
- Διαγραφή φίλου (**deletefromFriends**)
- Διαγραφή ποστ από το προφίλ του χρήστη (**delPost**)

Οι παρενθέσεις δίπλα από κάθε λειτουργία περιλαμβάνουν τα ονόματα των συναρτήσεων στην κλάση **user_service**.

3.2 Documentation

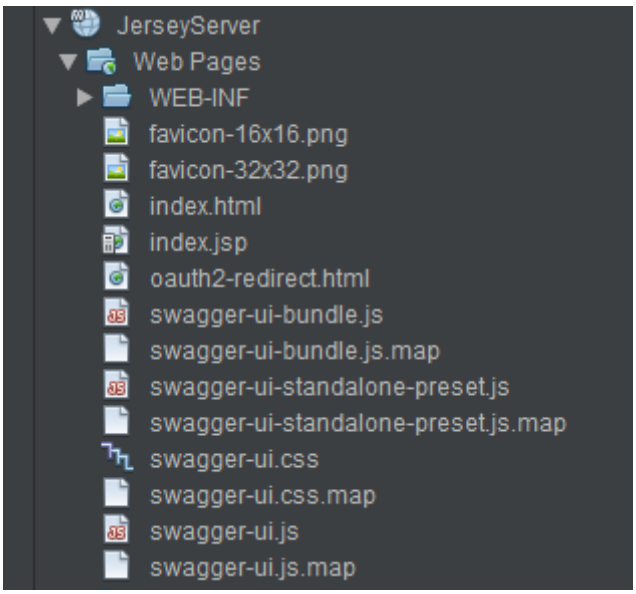
Για την δημιουργία του documentation έπρεπε να υλοποιήσουμε το μοντέλο Swagger. Στην ουσία είναι ένα εργαλείο που περιέχει οδηγίες για το πώς λειτουργεί η εφαρμογή μας. Τι εισόδους παίρνει, τι εξόδους δίνει καθώς και το είδος του αιτήματος (**HTTP Request Method**). Στην περίπτωση μας βοήθησε να ελέγχουμε την πορεία του project αφού ο πελάτης ολοκληρώθηκε τελευταίος. Το χρησιμοποιήσαμε για να φτιάχνουμε http request και να ελέγχουμε τι μας γυρνάει ο server ως απάντηση. Για την παραμετροποίηση του χρειάστηκε να κατανοήσουμε τι είναι τα annotations και πως αυτά λειτουργούν. Επίσης έπρεπε να παραμετροποιήσουμε το web.xml αρχείου που βρίσκεται στο φάκελο WEB-INF του project. Τα annotations που χρησιμοποιήθηκαν για τον ορισμό της μεθόδου αιτήματος ήταν **@PUT** & **@GET**. Παράλληλα χρησιμοποιήσαμε το **@Path** για τον καθορισμό του μονοπατιού όπου ο χρήστης θα καλέσει την αντίστοιχη λειτουργία. Τέλος προσθέσαμε και τα **@Produce** &

@Consume για να δείξουμε την παραγωγή/κατανάλωση πόρων που κάνει η κάθε λειτουργία. Με το path του λέμε στην ουσία το πώς θα προσπελάσει την κάθε λειτουργία ,και αυτή με την σειρά της θα καλέσει την υλοποιημένη συνάρτηση που βρίσκεται στον εξυπηρετητή. Επίσης του δώσαμε άλλα 3 annotations. Το **@Api** που το λέει το path που βρίσκονται όλες οι λειτουργίες και μετέπειτα προσθέτοντας το extra path της κάθε διαδικασίας φτάνεις σαυτήν. Ακόμη έχουμε το **@ApiResponses** που είναι wrapper του **@ApiResponse** και περιέχει τις δυνατές απαντήσεις του server. Τέλος έχουμε το **@RequestMapping** όπου κατευθύνει τα αιτήματα του πελάτη.

```
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>com.mycompany.jerseytest,io.swagger.jaxrs.listing</param-value>
<!--            <param-value>com.mycompany.testtee</param-value>-->
        </init-param>
        <load-on-startup>1</load-on-startup>

        </servlet>
        <servlet>
            <servlet-name>SwaggerBootstrap</servlet-name>
            <servlet-class>com.mycompany.jerseytest.SwaggerConfiguration</servlet-class>
              <load-on-startup>2</load-on-startup>
        </servlet>
```

Εικόνα 3.2.1 : Παραμετροποίηση web.xml αρχείου

Σαφώς μην ξεχάσουμε ότι ήταν απαραίτητο το κατέβασμα του swagger-ui μέσω GitHub.



Εικόνα 3.2.2 : Path αρχείων swagger-ui

```
@Path("user/")
@Api(value = "/", description = "Operations")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "OK"),
    @ApiResponse(code = 403, message = "Permission denied"),
    @ApiResponse(code = 500, message = "Something went wrong"),})
@RequestMapping(method = RequestMethod.GET, value = "/")
public class FirstClass {
```

Εικόνα 3.2.3 : Annotation που αφορούν όλες τις διαδικασίες

3.3 Screens Κώδικα FirstClass.java

```
//Αναζήτηση φίλων
@GET
@Path("/searchmyfriends/{id}")
@Produces({MediaType.APPLICATION_JSON})
@Consumes({MediaType.APPLICATION_JSON})
public Response searchFriends(@PathParam("id") String id) {
    List<String> ll;
    ll = user.searchMyFriends(id);
    return user.getRes2(ll);
}
```

Εικόνα 3.3.1 : Διαδικασία αναζήτησης φίλων

```
//Διαγραφή ποστ
@PUT //desc->περιεχόμενο post που θα διαγράψει
@Path("/deletepost/{id}&{desc}")
@Produces({MediaType.APPLICATION_JSON})
@Consumes({MediaType.APPLICATION_JSON})
public Response deletePost(@PathParam("id") String id, @PathParam("desc") String desc) {
    message = user.delPost(id, desc);
    return user.getRes(message);
}
```

Εικόνα 3.3.2 : Διαδικασία διαγραφής ποστ

Τα πεδία μέσα στις αγκύλες αναπαριστούν τις μεταβλητές που θα καταχωρήσει ο χρήστης με το αίτημα του. Πέραν από αυτές τις δύο μεθόδους δημιουργούνται και άλλες που δεν επισυνάπτονται για λόγους λακωνικότητας και ευαναγνωστικότητας της αναφοράς. Από όλες τις διαδικασίες έχουμε δύο τρόπους επιστροφής.

```
public Response getRes(Boolean message)
{
    if (message) {
        return Response.status(Response.Status.OK).build();
    } else {
        return Response.status(Response.Status.FORBIDDEN).build();
    }
}
```

Εικόνα 3.3.3 : Πρώτη επιστροφή

```java
public Response getRes2(List<String> ll){

    if (ll == null) {
        return Response.status(Response.Status.FORBIDDEN).build();
    } else {
        GenericEntity generic = new GenericEntity<List<String>>(ll) {
        };
        return Response.status(Response.Status.OK).entity(generic).build();
    }
}
```

Εικόνα 3.3.4 : Δεύτερη επιστροφή

Με τις δύο αυτές μεθόδους μειώσαμε υπερβολικά την επαναχρησιμοποίηση κώδικα. Και στις δύο περιπτώσεις το Response αφορά status ,απλά στην δεύτερη περίπτωση το στάτους εξαρτάτε από μία μεταβλητή τύπου **GenericEntity**. Αυτού του είδους η επιστροφή πραγματοποιείτε από τις διαδικασίες που επιστρέφουν λίστα, διότι παρόλο που το entity παίρνει ως παράμετρο Object type , δεν λειτουργούσε με την List.

3.4 Υλοποίηση User_Service.java

Η κλάση αυτή περιέχει των κώδικα υλοποίησης των συναρτήσεων που καλούνται από την FirstClass.java μέσω του Path. Παραθέτουμε τα στιγμιότυπα των μεθόδων της ενότητας 3.3 .

```java
//Αναζήτηση φίλων βάση id συνδεδεμένου χρήστη
public List<String> searchMyFriends(String id) {
    ArrayList<String> ll = new ArrayList();

    try {
        st = (Statement) connectToDB();

        String sql = "SELECT u_id1,u_id2 FROM friends WHERE '" + id + "' = u_id1 OR '" + id + "' = u_id2";

        set = st.executeQuery(sql);

        while (set.next()) {
            String user1 = set.getString("u_id1");
            String user2 = set.getString("u_id2");

            //Επειδή στη βάση θα υπάρχουν και αντίστροφες εγγραφές
            //ελέγχω αν με έχει κάνει add ο άλλος ή εγώ αυτόν
            if (id.equals(user1)) {
                ll.add(user2);
            } else if (id.equals(user2)) {
                ll.add(user1);
            }
        }
    } catch (SQLException ex) {
        Logger.getLogger(user_service.class.getName()).log(Level.SEVERE, null, ex);
    }
    return ll;
}
```

Εικόνα 3.4.1 : Μέθοδος αναζήτησης φίλων – πλευρά server

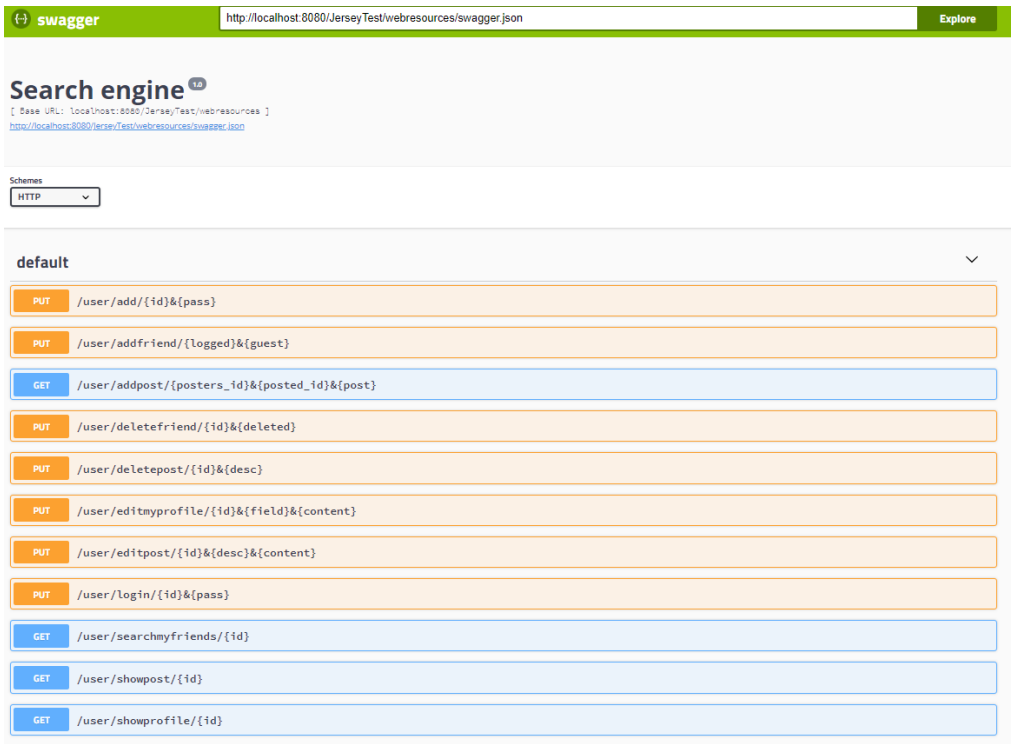
```
//Διαγραφή post βάση id και περιεχομένου
public boolean delPost(String id, String desc) {

    try {
        st = (Statement) connectToDB();
        String sql1 = "select p_id from profile where u_id = '" + id + "'";
        set = st.executeQuery(sql1);
        if (set.next()) {
            String sql = "delete from posts where p_id=" + set.getInt("p_id") + " AND description='" + desc + "'";
            st.execute(sql);
            message = true;
        }

    } catch (SQLException ex) {
        Logger.getLogger(user_service.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        return message;
    }
}
```

Εικόνα 3.4.2 : Μέθοδος διαγραφής ποστ από το προφίλ – πλευρά server

Όπως παρατηρούμε οι μέθοδοι παίρνουν ως εισόδους κάποιες παραμέτρους. Οι παράμετροι αυτοί εκχωρούνται από τον χρήστη με την υποβολή του αιτήματος. Οι παράμετροι αναπαρίστανται με «**{όνομα παραμέτρου}**». Κατόπιν δημιουργούμε τα κατάλληλα queries και υλοποιούμε ότι ζητείτε.

3.5 Swagger – UI

Μετά την ολοκλήρωση λοιπόν των απαραίτητων μεθόδων και παραμετροποιήσεων στο web.xml καθώς και την πρόσθεση των αρχείων του UI πρέπει να παίρνουμε το αποτέλεσμα που φαίνεται στην παρακάτω οθόνη.



Εικόνα 3.5.1 : Swagger - UI

4 Δημιουργία Client

Για την δημιουργία της εφαρμογής πελάτη δημιουργήσαμε Maven Project και συγκεκριμένα Java application. Χρησιμοποιήσαμε τη βιβλιοθήκη swing για να φτιάξουμε το γραφικό περιβάλλον. Προσθέσαμε επίσης τα απαραίτητα dependencies που βρήκαμε ότι χρειάζονται από το 11ο εργαστηριακό φύλλο. Η εφαρμογή του πελάτη αποτελείται από τέσσερα αρχεία. Τα τρία αφορούν το γραφικό περιβάλλον. Συγκεκριμένα το **MyRestFram.java** αφορά το **JFrame** της εφαρμογής και τα **LoginPanel.java** , **ProfileGUI.java** είναι τα **JPanel**. Το LoginPanel εκχωρείτε στο JFrame μόνο κατά το άνοιγμα της εφαρμογής όπου ζητούνται στοιχεία αυθεντικοποίησης. Αφού κάνει είσοδο ή εγγραφή ο χρήστης μεταφέρεται στην άλλη σελίδα του ProfileGUI. Στην ουσία γίνεται εκχώρηση αυτού στο frame μέσω της συνάρτησης switch11() που έχουμε υλοποιήσει. Το τέταρτο αρχείο αφορά το back-end της εφαρμογής του client. Τον τρόπο επικοινωνίας του δηλαδή με τον εξυπηρετητή και είναι το Leitourgies.java.

4.1 Leitourgies.java

Για την ωραία δομή και ροή της αναφοράς επισυνάπτουμε τις οθόνες των δύο μεθόδων που εξετάζουμε σαυτήν.

```
public Leitourgies() { //Αρχικοποίηση Client's configuration
    ClientConfig config = new DefaultClientConfig();
    config.getClasses().add(JacksonJaxbJsonProvider.class);
    client = Client.create(config);
}
```

Εικόνα 4.1.1 : Παραμετροποίηση πελάτη

```
//Ανάκτηση λίστας φίλων
public List getFriends(String user) { //Ορισμός path που θα βρει την υλοποίηση + user ,για τον οποίο θα την τρέξει
    WebResource target = client.resource("http://localhost:8080/JerseyTest/webresources/user/searchmyfriends/" + user);
    ClientResponse res = target.type("application/json").get(ClientResponse.class); //Ορισμός τύπου του resource και μεθόδου του αιτήματος

    if (res.getStatus() != 200) {
        System.out.println("Something went wrong");
    } else { //Εκχώρηση του αποτελέσματος σε GenericType List πρίν την επιστροφή
        result = res.getEntity(new GenericType<List<String>>() {
        });
    }
    return result;
}
```

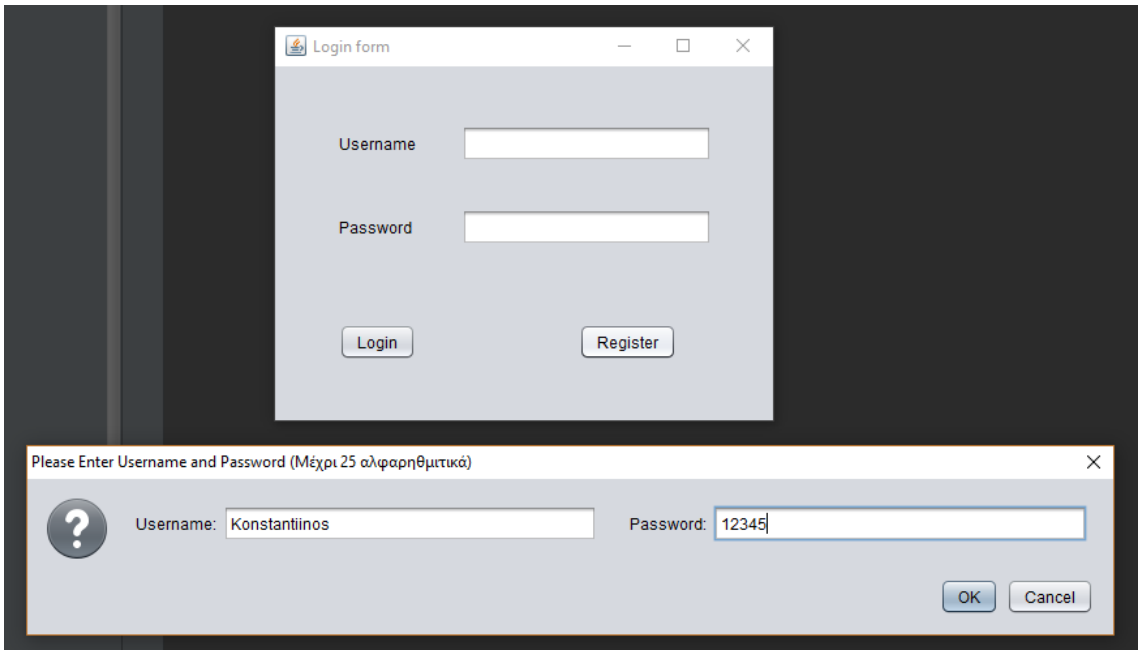
Εικόνα 4.1.2 : Μέθοδος αναζήτησης φίλων – πλευρά Client

```
//Διαγραφή post
public String deletePost(String id, String post) {
    String desc = post.substring(post.indexOf("posted : ") + 9);
    String replace = desc.replace(" ", "%20");
    String input = id + "&" + replace;
    WebResource target = client.resource("http://localhost:8080/JerseyTest/webresources/user/deletepost/" + input);
    ClientResponse res = target.type("application/json").put(ClientResponse.class);

    if (res.getStatus() != 200) {
        message = "Something went wrong";
    } else {
        message = "Post Deleted";
    }
    return message;
}
```

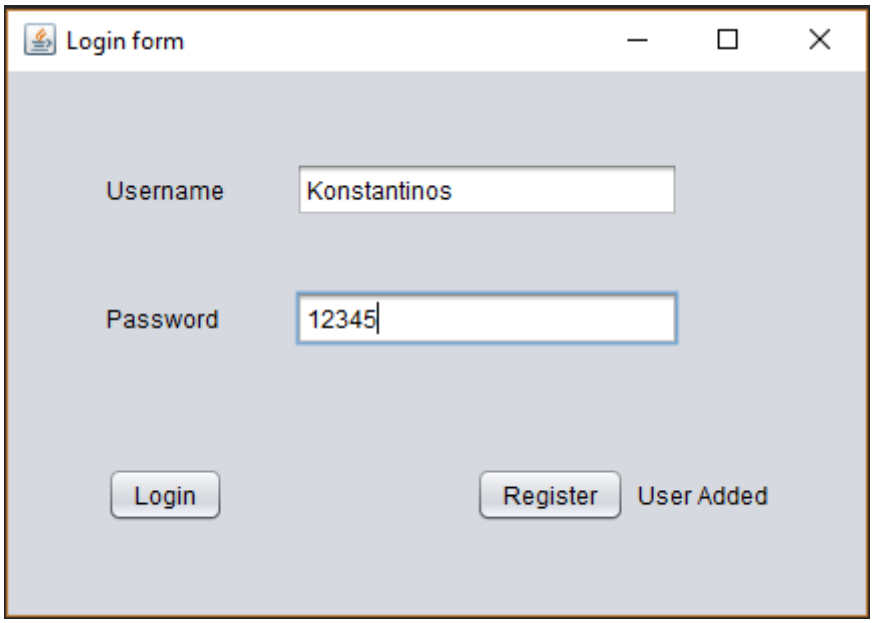
Εικόνα 4.3 : Μέθοδος διαγραφής post – πλευρά Client

Όπως βλέπουμε και στην πλευρά του πελάτη σε κάθε μέθοδο του ορίζουμε το path όπου περιέχεται ο πόρος που θα προσπελάσει για να του επιφέρει εντέλει αποτέλεσμα. Οι κλάσεις που είναι σημαντικές για αυτήν την υλοποίηση είναι οι **WebResource** & **ClientResponse**. Ανάλογα το αποτέλεσμα που θα γυρίσει ο σέρβερ στην μεταβλητή res (ClientResponse class) καταλαβαίνουμε αν πραγματοποιήθηκε το αίτημα ή αν πήγε κάτι στραβά. Συνεπώς η ροή ολόκληρης της υπηρεσίας γίνεται ως έχει. Αρχικά ο client ανοίγει την εφαρμογή και συνδέετε. Πατώντας Register ή Login καλεί με το path που έχουμε ορίσει στο WebResource του Login/Register τον αντίστοιχο πόρο στην κλάση FirstClass.java. Έπειτα από κει τρέχει η μέθοδος και καλεί την πραγματική μέθοδο ,που βρίσκεται στην κλάση user_service, που θα γυρίσει αποτέλεσμα στην μέθοδο της FirstClass και αυτή με την σειρά της στο πελάτη.

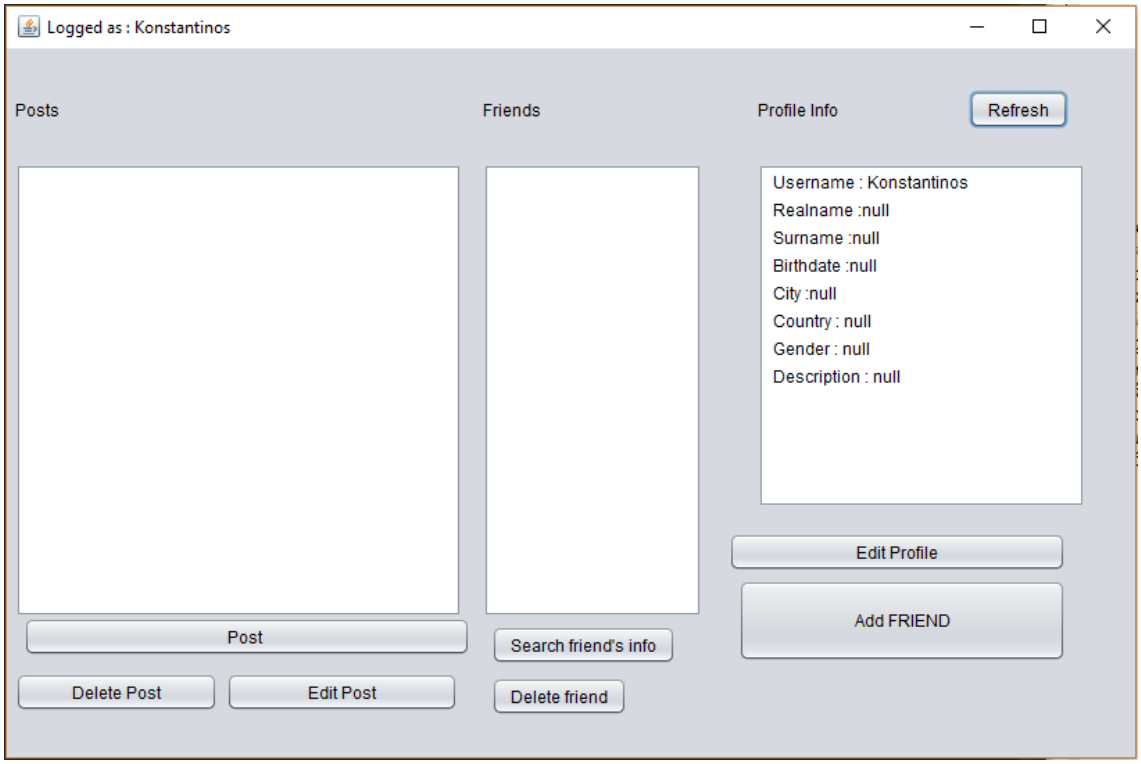
5 Οθόνες εκτέλεσης λειτουργίας εγγραφής & εισόδου



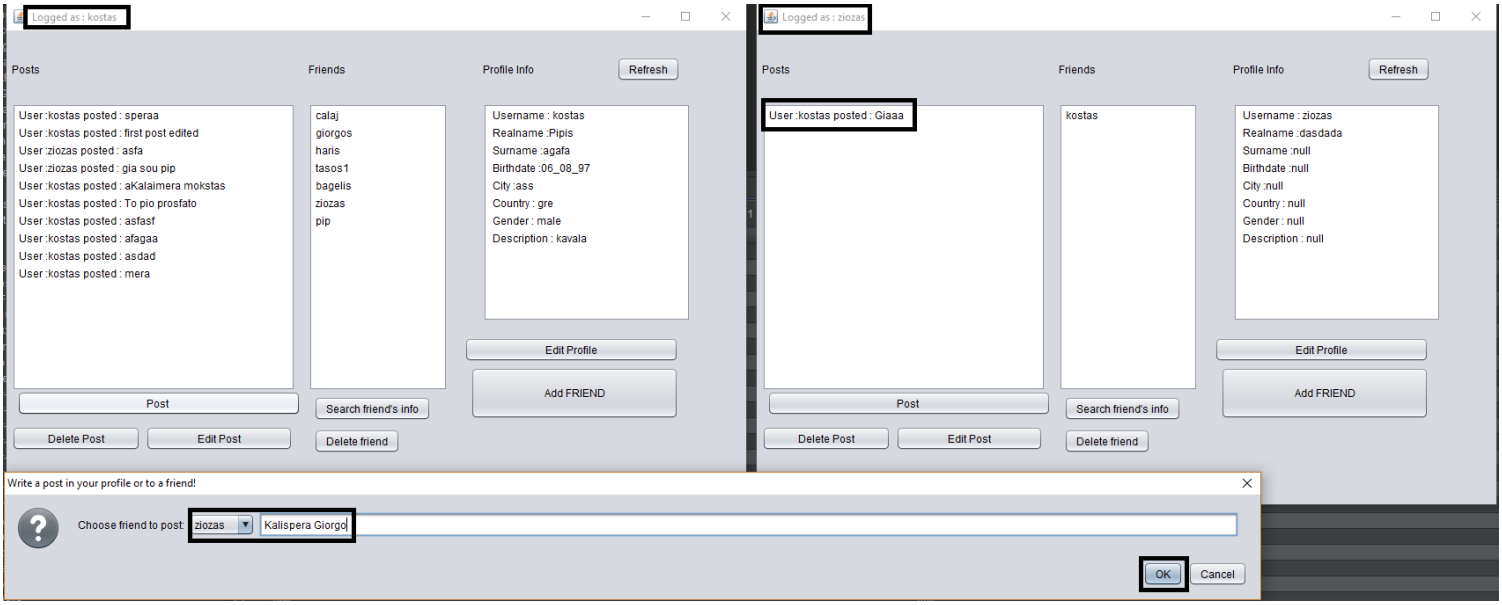
Εικόνα 5.1 : Δημιουργία χρήστη (Username:Password) Konstantinos : 12345



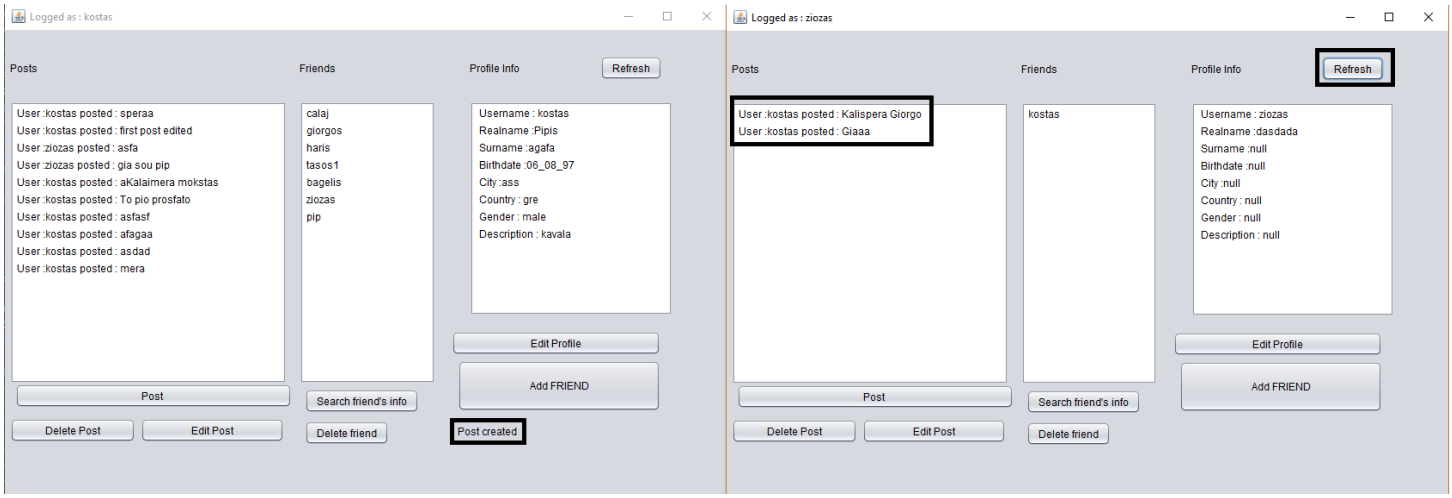
Εικόνα 5.2 : Είσοδος με τον χρήστη που φτιάξαμε προηγουμένως



Εικόνα 5.3 : ProfileGUI JPanel



Εικόνα 5.4 : Live Post από χρήστη : kostas σε χρήστη : ziozas



Εικόνα 5.5 : Έλεγχος post

Για την επικύρωση ότι το σύστημα μας λειτουργεί ως κατανεμημένο ανοίγουμε δύο client και στέλνουμε αίτημα από τον client_1 που το αποτέλεσμα θα φανεί στον Client_2. Για την ακρίβεια κάνουμε ένα post με τον χρήστη kostas στο profile του χρήστη ziozas.

6 Συμπέρασμα

Συνοψίζοντας έχουμε καταφέρει να υλοποιήσουμε μία Restful web εφαρμογή η οποία συγχρονίζεται άμεσα και λειτουργεί ως κατανεμημένη. Κατανοήσαμε έννοιες όπως : πόρος, μονοπάτι, ετικέτα, αίτημα, παρακολούθηση αιτημάτων κ.ά. Εξετάσαμε την λειτουργία του swagger και δημιουργήσαμε ένα documentation για πρώτη φορά. Τέλος παραμετροποιήσαμε το δικό μας API.

7 Βιβλιογραφία

~mkyong. “RESTful Java client with Jersey client”. July 19th, 2011. Link.

~CodeJava.net. “JComboBox basic tutorial and examples”. 9th April 2017. Link.

~Google. “ASCII Table”. Link.

~Java2Novice. “Input json request with Jersey and Jackson”. Link.

~mkyonk. “JSON example with Jersey + Jackson”. July 19th, 2011. Link.

~techonthenet. “Oracle / PLSQL: UPDATE Statement”. Link.

~icancode. “Configuring Swagger with Jersey”. May 17th, 2017. Link.

~ neocorp. “Swagger rest api documentation pom.xml”. April 6th, 2017. Link.

~W3Schools. “HTTP Request Methdos”. Link.