ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΔΙΚΤΥΑ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ-ΠΡΩΤΟ ΠΑΡΑΔΟΤΕΟ 2018-2019

KEEP
CALM
IT WORKS
ON
MY MACHINE

ΟΜΑΔΑ 4

Ον/μο: Σωκράτης Μπέης **ΑΜ:** 1115201300113

Ον/μο: Συμέλα-Φωτεινή Κομίνη **ΑΜ:** 1115201400072

Ον/μο: Ευάγγελος Τζιρώνης **ΑΜ:** 1115201400199

Ον/μο: Ιωάννης Μανωλάκης **ΑΜ:** 1115201500088

ANDROID

Η επικοινωνία android-edge επιτυγχάνεται μέσω του Mqtt_broker και τη χρήση τεσσάρων topics(mobile1,mobile2,edge1 και edge2). Το κινητό Χ κάνει publish το μήνυμα που περιγράφει η εκφώνηση, στο topic «mobileX» και subscribe στο topic «edgeX». Ομοίως, ο edge server κάνει subscribe στο topic «mobileX», ώστε να λάβει το μήνυμα από το κινητό Χ και publish την κατάλληλη απάντηση στο topic «edgeX».

Τα βασικά αρχεία που απαρτίζουν την εφαρμογή **«DiT 404»** είναι τα:

- **Layout-xml**
  - **activity_main.xml :** Αποτελεί την κεντρική οθόνη της εφαρμογής. Περιέχει τα δύο κουμπιά connect και disconnect για σύνδεση και αποσύνδεση αντίστοιχα, καθώς και το menu για έξοδο από την εφαρμογή και ρύθμιση της συχνότητας αποστολής csv αρχείων.
  - **activity_settings.xml :** Αποτελεί την οθόνη επιλογής της συχνότητας απόστολης csv αρχείων, ανά 5, 10(default) ή 20 seconds.
- **JAVA**
  - **MainActivity.java**
  - **SettingsActivity.java**
  - **Connect.java**
  - **Disconnect.java**

Κατά την εκκίνηση της εφαρμογής ζητούνται από τον χρήστη permissions για την κάμερα, για ανάγνωση της sdcard και για λήψη της τοποθεσίας του κινητού.

ΕΝΟΤΗΤΑ 1

File: **MainActivity.java**

Με το που ξεκινάει η εφαρμογή καλείται η **onCreate()**. Σε εκείνη καλούνται οι συναρτήσεις **init()** και **restoreValuesFromBundle(),** ζητείται permission για την κάμερα, αρχικοποιούνται οι κατάλληλες μεταβλητές-συναρτήσεις για τη σύνδεση/επικοινωνία με τον Mqtt_broker και καλείται το **asyncTask Connect.java**.

- **init():** Αρχικοποίηση των μεταβλητών που χρησιμοποιούνται στο MainActivity και αφορούν το GPS, το WiFi, το accelerometer και άλλες χρήσιμες μεταβλητές για την ομαλή λειτουργία της εφαρμογής.

- **restoreValuesFromBundle():** Χρησιμοποιείται για την επανάκτηση μεταβλητών που αφορούν το GPS σε περίπτωση που καταστραφεί το activity λόγω περιστροφής οθόνης.

- **Connect.java:(βλέπε ΕΝΟΤΗΤΑ 3).**

Αφού κληθεί η **Connect.java**, γίνει επιτυχώς η σύνδεση με τον Mqtt_broker και το subscribe στο αντίστοιχο topic (edge1/edge2) καλείται η συνάρτηση **startLocationUpdates()** για τη συνεχή ενημέρωση της τοποθεσίας του κινητού.

- **startLocationUpdates():** Ελέγχεται αν έχουν δωθεί από τον χρήστη τα permissions για την ενεργοποίηση και τη χρήση του GPS. Αν έχουν δωθεί, τότε καλείται η συνάρτηση **publish**(), εναλλακτικά ο

χρήστης παραπέμπεται στις ρυθμίσεις του κινητού που αφορούν το GPS, ώστε να τα ενεργοποιήσει.

- **publish() :** Τρέχει μόνο εφόσον το κινητό έχει λάβει τουλάχιστον μία φορά την τοποθεσία του και ζητάει permissions για να διαβάσει το directory με τα csv αρχεία από την sdcard. Στην περίπτωση που έχουν δωθεί, φτιάχνει μία λίστα με όλα τα αρχεία που υπάρχουν στο directory και καλούνται οι συναρτήσεις **read_file()** και **makeMessageToPublish().**

- **read_file():** Τυχαία επιλέγεται ένα από τα αρχεία και όλο το περιεχόμενό του μετατρέπεται σε ένα string.

- **makeMessageToPublish():** Δημιουργείται το μήνυμα που θα κάνει publish το κινητό, στο κατάλληλο topic (mobile1/mobile2), με όλα τα δεδομένα που ζητάει η εκφώνηση. Να αναφερθεί ότι οι τιμές από το accelerometer ανανεώνονται και αποθηκεύονται σε ένα ArrayList κάθε φορά που αλλάζει η κατεύθυνση ή/και η κλίση του κινητού.

Σε οποιαδήποτε περίπτωση που η MainActivity μπει στο παρασκήνιο, καλείται η **stopLocationUpdates()**, στην οποία σταματούν οι ενημερώσεις για τη θέση του κινητού.

Σε περίπτωση που ο χρήστης τερματίσει την εφαρμογή, καλείται το **asyncTask Disconnect.java (βλέπε ΕΝΟΤΗΤΑ 4)**.

ΕΝΟΤΗΤΑ 2

File: **SettingsActivity.java**

Όταν ο χρήστης επιλέγει να αλλάξει συχνότητα από το menu, καλείται η **stopLocationUpdates()**, και με την επιστροφή από τη SettingsActivity καλείται ξανά η **startLocationUpdates()** με τη νέα συχνότητα ενημέρωσης.

Στην **onCreate()** , ανάλογα με τη συχνότητα που επιλέγει ο χρήστης, αλλάζει η συχνότητα με την οποία ενημερώνεται η θέση του κινητού και κατ' επέκταση η συχνότητα με την οποία γίνεται το publish.

Connect.java & Disconnect.java ως **AsyncTasks**

Οι κλάσεις **Connect** και **Disconnect** αποτελούν επέκταση της κλάσης **AsyncTask** , δηλαδή, οι ενέργειες που πραγματοποιούνται σε αυτές γίνονται σε ξεχωριστό από το main UI thread. Με αυτό τον τρόπο, δεν παρεμποδίζεται η λειτουργία της εφαρμογής.

ΕΝΟΤΗΤΑ 3

File: **Connect.java – AsyncTask**

Ξεκινώντας, ελέγχει αν είναι ανοιχτό το WiFi, και αν όχι, το ενεργοποιεί. Έπειτα, γίνεται το connection με τον broker και σε περίπτωση επιτυχίας το κινητό κάνει subscribe στο αντίστοιχο topic (edge1/edge2) ζητώντας τα κατάλληλα permissions για το GPS.

ΕΝΟΤΗΤΑ 4

File: **Disconnect.java – AsyncTask**

Κάνει disconnect από τον broker και σε περίπτωση επιτυχίας καλείται και η **stopLocationUpdates().**

EDGE SERVER

Η **επικοινωνία edge-backhaul** επιτυγχάνεται μέσω web sockets. Ο edge δημιουργεί ένα socket περνώντας ως παραμέτρους τον αριθμό θύρας που έχει επιλέξει ο backhaul και την IP address του backhaul. Έπειτα, μέσω της **read()** διαβάζει όλα τα bytes του csv αρχείου που του στέλνει ο backhaul στο socket. Αφού διαβάσει όλο το αρχείο, κλείνει το socket. Εν συνεχεία, χρησιμοποιώντας την κλάση CSVReader της βιβλιοθήκης opencsv διαβάζει το αρχείο csv και το μετατρέπει σε ένα ArrayList. Αφού γίνει αυτό, για κάθε γραμμή-πείραμα του csv αρχείου, αποθηκεύει τοπικά σε ένα αντικείμενο της κλάσης Experiment το όνομα του πειράματος και το feature vector του, όπως ζητείται στην εκφώνηση.

Για την **επικοινωνία edge-android,** ο edge συνδέεται στον Mqtt_broker και κάνει subscribe στα topics «mobile1» και «mobile2», με σκοπό τη λήψη μηνυμάτων απο τα δύο κινητά. Κάθε φορά που λαμβάνει ένα μήνυμα, εξετάζει το topic αυτού, και κάνει publish ένα τυχαίο (για την πρώτη φάση του πειράματος) αριθμό απο 0 έως 2, στο κατάλληλο topic (edge1 ή edge2).

Training Set Creation

(Final CSV creation to send to the Edge Server by the Backhaul Server)

Files: **CSV_Reader.java**

Experiment.java

Το αρχείο **Experiment.java** είναι μία κλάση, κάθε αντικείμενο της οποίας αντιστοιχεί σε μία γραμμή του τελικού CSV. Περιέχει πέρα από τα απαραίτητα πεδία και τον constructor και δύο συναρτήσεις επιστροφής των πεδίων αυτών**: public String get_name()** και **public double[] get_feature_vector().**

Στο αρχείο **CSV_Reader.java**:

String OUT_CSV_PATH : Το τελικό αρχείο που προκύπτει από την εκτέλεση.

String CSV_FILE_PATH1 : Το path του φακέλου όπου περιέχονται τα πειράματα για την επεξεργασία τους.

Δημιουργούμε μια μεταβλητή folder για την επεξεργασία όλου του φακέλου με τα CSV αρχεία του training.

Στη συνέχεια φτιάχνουμε μία λίστα που περιέχει όλα τα pathnames των CSV αρχείων με τα πειράματα καθώς και ένα πίνακα με αντικείμενα κλάσης Experiment (**exp_array**), μεγέθους όσο και ο αριθμός των πειραμάτων.

Διατρέχουμε τον φάκελο με τα CSV και ενόσω υπάρχει διαθέσιμο αρχείο, το διαβάζουμε χρησιμοποιώντας την κλάση **CSVReader** του com.opencsv . Δημιουργείται η λίστα records που περιέχει όλα τα δεδομένα σε μορφή String του εξεταζόμενου CSV (πίνακες από Strings).

Αποθηκεύουμε τις τιμές κάθε στήλης-καναλιού σε ένα vector

input_vectors[i] , αφού μετατρέψουμε τα στοιχεία Strings σε Double και μετά υπολογίζουμε την εντροπία για το συγκεκριμένο κανάλι με τη συνάρτηση **Entropy.calculateEntropy()** με όρισμα το **input_vectors[i]** και για τα 14 κανάλια.

Επίσης, για κάθε στοιχείο του **exp_array** , καλείται ο αντίστοιχος constructor.

Τέλος, χρησιμοποιώντας τη κλάση **CSVWriter** του com.openscv, γράφουμε ανά γραμμή το αντίστοιχο όνομα πειράματος και το feature vector με τις υπολογισμένες εντροπίες όπως ζητείται στην εκφώνηση δημιουργώντας το τελικό CSV που θα στείλουμε στον Edge Server.

Backhaul Server

(Connection to Edge Server Section)

File: **Backhaul.java**

Αρχικά, επιλέγουμε έναν αριθμό θύρας για την σύνδεση. Δημιουργούμε ένα socket στη συγκεκριμένη θύρα και αναγκάζουμε τον Backhaul να αναμένει **συνεχώς** μήνυμα σύνδεσης από τον Edge Server. Μόλις επιτευχθεί σύνδεση, ο Backhaul δημιουργεί ένα πίνακα από bytes, μεγέθους όσο το τελικό CSV αρχείο που προέκυψε από την εκτέλεση του **CSV_Reader.java,** και το προσθέτει στο stream για να το διαβάσει ο Edge. Αύτο επιτυγχάνεται με τη χρήση των εντολών:

- **read(mybytearray,0,mybytearray.length),** η οποία διαβάζει όλα τα bytes του mybytearray.
- **getOutputStream(),** η οποία δημιουργεί ένα output stream του Backhaul για να γράψει στο συγκεκριμένο socket.
- **write(mybytearray,0,mybytearray.length),** η οποία γράφει στο μόλις εγκαθιδρυμένο stream όλα τα bytes από τον προηγούμενο πίνακα.
- **flush(),** η οποία κλείνει το stream.
- **close(),** η οποία κλείνει το socket.

JDBC

(Connection to the ‘project_db’ Database by the Backhaul)

File: **DB_Conn.java**

---

project_db_model.mwb: Είναι το μοντέλο της βάσης για το παραδοτέο

project_db_events.sql: Είναι ο κώδικας για τη δημιουργία της βάσης. Έχει δημιουργηθεί και μία «dummy» πλειάδα για έλεγχο με την check_data().

Γίνεται χρήση του **mysql-connector-java-8.0.13.jar** για τη σύνδεση στον SQL server από Windows 10.(Περιέχεται στο φάκελο JDBC_New)

Χρησιμοποιούμε τον jdbc Driver για τη σύνδεση στη βάση μας. Κάνουμε τους απαραίτητους ελέγχους για το αν είναι εγκατεστημένος πριν προχωρήσουμε στη σύνδεση. Καλούμε την συνάρτηση **DriverManager.getConnection()** με ορίσματα το όνομα της βάσης, το αντίστοιχο port, το username και το password του δημιουργού για να συνδεθούμε. Μετά από τους ελέγχους για ύπαρξη σύνδεσης, έχουμε δημιουργήσει μία βοηθητική συνάρτηση **check_data(),** για να εξάγουμε δεδομένα από τη βάση με ένα select statement. Η χρήση της είναι καθαρά για τον έλεγχο ορθής λειτουργίας του κώδικα (μπορεί να παραληφθεί στο πρώτο παραδοτέο).