FoodApp

Μάθημα: Κατανεμημένα Συστήματα

Μέλη Ομάδας:

- **Κυριακόπουλος Θεόδωρος – p3190098**
- **Γκριτζέπης Αλέξανδρος – p3190278**
- **Γιαννίκου Ευσταθία – p3190035**

Το **FoodApp** είναι ένα κατανεμημένο σύστημα παραγγελίας φαγητού που αναπτύχθηκε σε Java και Kotlin και λειτουργεί με αρχιτεκτονική τύπου Master–Worker. Το σύστημα χρησιμοποιεί MapReduce για αποδοτική επεξεργασία και συγχρονισμό δεδομένων, καθώς και Android εφαρμογή για φιλικό περιβάλλον χρήστη. Το σύστημα διαχειρίζεται καταστήματα, προϊόντα, αναζητήσεις και αγορές, ενώ χρησιμοποιεί πρωτόκολλο TCP για την επικοινωνία.

---

1. Αρχιτεκτονική Συστήματος

Το σύστημα αποτελείται από δύο βασικά μέρη:

1. **Backend (Java - Desktop)**: Υλοποιημένο σε Java, περιλαμβάνει διαχείριση καταστημάτων, αναζητήσεις και αγορές μέσω TCP sockets.
   - MasterServer
   - WorkerServer (3 instances)
   - ManagerApp (CLI)
   - ClientApp (CLI)

2. **Frontend (Android)**: Εφαρμογή Android σε Kotlin με Jetpack Compose για απλή και λειτουργική διεπαφή χρήστη.
   - Mobile App (Jetpack Compose)
   - TcpClient (για σύνδεση με τον Master)

Η επικοινωνία γίνεται μέσω TCP sockets. Όλα τα δεδομένα διακινούνται ως μορφοποιημένα μηνύματα (SEARCH:..., BUY:..., ADD_STORE_DATA:...), και η αρχιτεκτονική ακολουθεί κατανεμημένη λογική με MapReduce pattern.

---

2. Backend (Java)

2.1 Master Server

Ο MasterServer:

- Ακούει σε συγκεκριμένη πόρτα (π.χ., 5000)

- Αποδέχεται συνδέσεις από Workers, Clients και Android
- Διατηρεί λίστα Workers και προωθεί σε αυτούς αιτήματα

Λειτουργίες:

- Επιλέγει τον κατάλληλο Worker με βάση hash(storeName) % workerCount
- Συλλέγει αποτελέσματα από όλους τους Workers
- Κάνει reduce των αποτελεσμάτων και στέλνει πίσω στον Client ή Android

2.2 Worker Server

Κάθε Worker:

- Αποθηκεύει τοπικά καταστήματα (Store objects)
- Εκτελεί filtering queries (SEARCH, BUY, QUERY)
- Υποστηρίζει synchronized σε critical sections (read/write σε storeMap, αποθέματα, πωλήσεις)

Υποστηριζόμενες εντολές:

- ADD_STORE_DATA:{json}
- ADD_PRODUCT:store:product
- REMOVE_PRODUCT:store:product
- SEARCH:category,min,max,price,distance,lat,lon
- BUY:store:product:quantity
- QUERY_SALES_BY_CATEGORY, QUERY_SALES_BY_PRODUCT

2.3 ManagerApp

Ο CLI client για προσθήκη καταστημάτων και προϊόντων.

Παράδειγμα εντολής:

ADD_STORE_JSON:src/data/allo.json

ADD_PRODUCT:MyStore:Pizza Fun,pizza,10,5.99

2.4 ClientApp

CLI εφαρμογή για αναζήτηση και αγορά:

SEARCH:burgers,2,5,$,5.0,37.9838,23.7275

BUY:Pasta Mama:Napolitana:2

---

3. MapReduce Λογική και synchronized

Η λογική Map–Reduce υλοποιήθηκε με τρόπο ώστε η αναζήτηση να επεξεργάζεται τοπικά στους Workers (Map), και η ενοποίηση να γίνεται αποκλειστικά στον Master (Reduce). Για τις συναλλαγές τύπου αγοράς (BUY), εφαρμόστηκε hash κατανομή του store ώστε κάθε Worker να έχει αποκλειστική ευθύνη για συγκεκριμένα καταστήματα, επιτρέποντας έτσι κατανεμημένη διαχείριση αποθεμάτων. Για την αποφυγή συγκρούσεων, χρησιμοποιήθηκαν synchronized blocks σε όλα τα κρίσιμα shared δεδομένα.

MapReduce Λογική

- Η κατανομή των δεδομένων γίνεται σε πολλαπλούς Workers μέσω JSON store εισαγωγών.
- Η επεξεργασία γίνεται τοπικά σε κάθε Worker (Map).
- Ο Master ενώνει τα αποτελέσματα με βάση το αίτημα (Reduce), διατηρώντας τη σωστή σειρά και πλήρη πληροφόρηση.
- Η αρχιτεκτονική υποστηρίζει ευκολία κλιμάκωσης (π.χ. προσθήκη επιπλέον Workers με αλλαγή στο config.txt).

Αναζήτηση (SEARCH)

- Ο Master, όταν λαμβάνει αίτημα τύπου SEARCH, το αποστέλλει σε όλους τους ενεργούς Workers.
- Κάθε Worker εφαρμόζει φίλτρα (Map) στα δικά του αποθηκευμένα καταστήματα (π.χ. κατηγορία, αστέρια, απόσταση, τιμή) και επιστρέφει μόνο τα αντίστοιχα.
- Ο Master συλλέγει τις απαντήσεις και τις συγχωνεύει (Reduce) σε μία τελική λίστα αποτελεσμάτων, την οποία στέλνει στον πελάτη ή στην Android εφαρμογή.

Αγορά (BUY)

- Για κάθε BUY:store:product:amount, ο Master εφαρμόζει κατανεμημένη επιλογή Worker με hash του storeName (Math.abs(storeName.hashCode()) % workers.size()), έτσι ώστε κάθε κατάστημα να ανήκει σε έναν συγκεκριμένο Worker.
- Το αίτημα αποστέλλεται μόνο σε αυτόν τον Worker, ο οποίος αναλαμβάνει την ενημέρωση αποθέματος, αύξηση συνολικών πωλήσεων και υπολογισμό εσόδων.

Συγχρονισμός (Synchronized)

Για την αποφυγή αγωνιστικών συνθηκών (race conditions), εφαρμόστηκε πλήρης συγχρονισμός:

- synchronized (storeMap): προστατεύει την πρόσβαση/τροποποίηση του χάρτη με τα καταστήματα.
- synchronized (store.getProducts()): προστατεύει τη λίστα προϊόντων κάθε καταστήματος κατά την προσθήκη, αφαίρεση ή αλλαγή αποθέματος.
- synchronized (store): για ταυτόχρονες ενημερώσεις στα totalSales και totalRevenue κάθε store.

---

4. Android Εφαρμογή

Η εφαρμογή αναπτύχθηκε με χρήση **Jetpack Compose** για το UI και **TCP socket** επικοινωνία με το backend. Υποστηρίζει πλήρως τις λειτουργίες SEARCH και BUY.

4.1 TcpClient (network/TcpClient.kt)

- Διαχειρίζεται τη σύνδεση TCP με τον Master Server.
- Ανοίγει Socket, BufferedReader και BufferedWriter.
- Στέλνει εντολές σε μορφή string (π.χ. SEARCH:..., BUY:...).
- Λαμβάνει απαντήσεις:
    - Πολλαπλές γραμμές που τελειώνουν με "END" (για SEARCH)
    - Μία γραμμή απάντησης (για BUY)
- Χειρίζεται σφάλματα σύνδεσης με fallback μήνυμα στον χρήστη.

4.2 Οθόνες UI (Jetpack Compose)

- **HomeScreen.kt**
    - Αρχική σελίδα με 2 πλήκτρα: "Αναζήτηση Καταστημάτων" και "Αγορά Προϊόντος"
    - Προστέθηκε εικόνα λογοτύπου και styled design
- **SearchScreen.kt**
    - Φόρμα εισαγωγής φίλτρων για αναζήτηση: κατηγορία, αστέρια, τιμή, τοποθεσία
- **ResultsScreenText.kt**
    - Προβολή των αποτελεσμάτων SEARCH ή απάντησης BUY
    - Περιέχει κουμπί επιστροφής στην αρχική
- **BuyScreen.kt**
    - Εισαγωγή καταστήματος, προϊόντος και ποσότητας
    - Στέλνει εντολή BUY: στον Master

4.3 Δεδομένα & Εργαλεία

- **SearchRequest.kt**
    - Data class για την αναπαράσταση των φίλτρων αναζήτησης
- **buildSearchString.kt**
    - Συνάρτηση που μετατρέπει το SearchRequest σε SEARCH: string
- **Καταστάσεις οθόνης (screenState)**

    - Το navigation μεταξύ οθονών υλοποιείται μέσω remember { mutableStateOf(...) }

- **Ασύγχρονη λειτουργία**
    - Όλες οι κλήσεις TCP γίνονται σε thread { ... } για να μην μπλοκάρει το UI thread

---

5. Χαρακτηριστικά

- Πλήρως κατανεμημένο backend σύστημα με **3 Workers**, που λειτουργούν παράλληλα.
- **Επεκτάσιμη αρχιτεκτονική**: Επιτρέπεται η προσθήκη επιπλέον Workers με ενημέρωση του αρχείου config.txt, χωρίς αλλαγές στον κώδικα του Master ή των Clients.
- Υλοποίηση **MapReduce λογικής** για την επεξεργασία εντολών SEARCH:
    - Κάθε Worker εκτελεί το δικό του μέρος του query (Map)
    - Ο Master συγκεντρώνει και συνδυάζει τα αποτελέσματα (Reduce)
- **Σύνδεση Android εφαρμογής** με τον Master Server μέσω **TCP sockets**, με πλήρως ασύγχρονη επικοινωνία.
- Χρήση synchronized blocks για **ασφαλή πρόσβαση σε κοινόχρηστα δεδομένα**, όπως:
    - Ο χάρτης των καταστημάτων (storeMap)
    - Η λίστα προϊόντων κάθε καταστήματος
    - Οι συνολικές πωλήσεις και έσοδα ανά κατάστημα

---

6. Οδηγίες Εκτέλεσης

6.1. Εκκίνηση Workers

Άνοιξε 3 ξεχωρισά terminal/cmd παράθυρα και για κάθε ένα:

>> javac worker/WorkerServer.java

>> java worker.WorkerServer <port>

Παράδειγμα:

>> java worker.WorkerServer 5001

>> java worker.WorkerServer 5002

>> java worker.WorkerServer 5003

6.2. Ρύθμιση Worker Config File

Άνοιξε ή δημιούργησε το αρχείο data/config.txt και εισήγαγε τις IP/Ports:

127.0.0.1:7001

127.0.0.1:7002

127.0.0.1:7003

6.3. Εκκίνηση Master Server

>> javac master/*.java

>> java master.MasterServer

Ο Master διαβάζει αυτόματα από config.txt τις διευθύνσεις των Workers.

6.4. Εκκίνηση Manager Console App

>> javac manager/ManagerApp.java

>> java manager.ManagerApp

Μέσα στην κονσόλα του Manager μπορείς να δώσεις εντολές όπως:

ADD_STORE_JSON:data/allo.json

ADD_PRODUCT:Pizza Fun:{"name":"Pizza Margherita","price":5.0,"stock":10}

6.5. Εκκίνηση Client Console App (προαιρετικά)

>> javac client/ClientApp.java

>> java client.ClientApp

Υποστηρίζει εντολές αναζήτησης και αγοράς, π.χ.:

SEARCH:burgers,1,5,$$,5.0,37.98,23.72

BUY:Pasta Mama:Napolitana:2

6.6. Εκκίνηση Android App (Frontend)

1. Άνοιξε το φάκελο FoodApp/ στο **Android Studio**
2. Στο TcpClient.kt, επιβεβαίωσε ότι η IP είναι 10.0.2.2 και port 5000 (ή ό,τι χρησιμοποιεί ο Master):

TcpClient("10.0.2.2", 5000)

3. Κάνε Sync with Gradle Files
4. Πάτησε **Run** για να εκκινήσεις στον emulator