Κατανεμημένα Συστήματα Εργαστήριο 1

Λέοναρντ Πέπα ics20033

Σχεδίαση Κατανεμημένου Συστήματος Τραπεζικών Λειτουργιών (ATM)

Αρχιτεκτονική Τριων Επιπέδων (3-Tier)

Επίπεδο Παρουσίασης

Είναι υπεύθυνο για την παρουσίαση των δεδομένων και την αλληλεπίδραση του χρήστη με το σύστημα, αυτό συνήθως επιτυγχάνεται με την χρήση γραφικής διασύνδεσης όπως είναι μία ιστοσελίδα τράπεζας, μία εφαρμογή web banking στο κινητό ή και το παραδοσιακό μηχάνημα ATM έξω από μία τράπεζα που σου παρέχει όλες τις επιλογές σε μια οθόνη.

Επίπεδο Επιχειρηματικής Λογικής

Είναι υπεύθυνο για όλη την λειτουργικότητα του συστήματος και την σύνδεση με την βάση δεδομένων. Δέχεται αιτήματα από τους χρήστες εκτελεί τους απαραίτητους ελέγχους, επεξεργάζεται τα αιτήματα και ανταποκρίνεται με το κατάλληλο μήνυμα για κάθε αίτημα. Στην περιπτωση μας θα μπορούσε να ελέγχει την αυθεντικοποίηση του χρήστη όταν θέλει να πραγματοποιήσει σύνδεση στο σύστημα ή αν διαθέτει επαρκή χρηματικό ποσό στον λογαριασμό του για την πραγματοποίηση ανάληψης κλπ. Η υλοποίηση του συγκεκριμένου επιπέδου γίνεται αναπτύσσοντας μια διεπαφή προγραμματισμού εφαρμογών (API) σε μία γλώσσα προγραμματισμού, που θα προσφέρει κάποιες υπηρεσίες όπως ειναι πχ η ανάληψη και η κατάθεση οι οποίες θα καταναλώνονται απο το επίπεδο παρουσίασης που είδαμε παραπάνω. Ένα παράδειγμα μιας διεπαφης θα μπορούσε να είναι ένα REST API που η επικοινωνία γίνεται με JSON αντικείμενα με το πρωτόκολλο HTTP.

Επίπεδο Δεδομένων

Είναι υπεύθυνο για την αποθήκευση και την διαχείριση των δεδομένων του συστήματος. Το συγκεκριμένο επίπεδο επικοινωνεί με το επίπεδο της επιχειρηματικής λογικής ώστε να γίνει εισαγωγή, ανάγνωση, τροποποίηση ή διαγραφή κάποιων δεδομένων. Η αποθήκευση συνήθως γίνεται σε μια βάση δεδομένων όπως η SQL. Στην περίπτωσή μας θα ήταν συνετό να χρησιμοποιήσουμε μια σχεσιακή βάση δεδομένων όπως η MariaDB για να μπορέσουμε απεικονίσουμε τις συσχετίσεις που πιθανόν να υπάρχουν στο σύστημα μας. Η Σύνδεση με την βάση θα μπορούσε να γίνει με κάποιον driver όπως είναι ο JDBC που μας επιτρέπει να συνδεθούμε και να αλληλεπιδράσουν με την βάση μέσω της γλώσσας προγραμματισμού java.

Συσχέτιση με το μοντέλο MVC (MODEL, VIEW, CONTROLLER)

- VIEW (Επίπεδο Παρουσίασης)
  - Login Page
  - Register Page
  - Main Page
- CONTROLLER (Επιχειρηματική Λογική)
  - Server
  - ControllerThread
  - ServerProtocol
  - RequestType
  - Request
  - response
- MODEL (Επίπεδο Δεδομένων)
  - Account
  - DB_Service
  - DB_Connector

Data Layer (MODEL)

Παράδειγμα ενός πίνακα SQL για τον λογαριασμό στο Τραπεζικό Σύστημα.

```
CREATE TABLE `account` (
  `id` int(11) NOT NULL,
  `pin` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `balance` double NOT NULL
);
```

Αναπαράσταση Πίνακα ώς οντότητα του συστήματος με μια κλάση

```
class Account {
    private int id;
    private int pin;
    private String name;
    private double balance;
}
```

Ψευδοκώδικας Υπηρεσιών της Βάσης Δεδομένων

Απλές λειτουργίες CRUD (CREATE, READ, UPDATE, DELETE)
Αν προκύψει το οποιοδήποτε λάθος τότε οι μέθοδοι επιστρέφουν null (τίποτα)

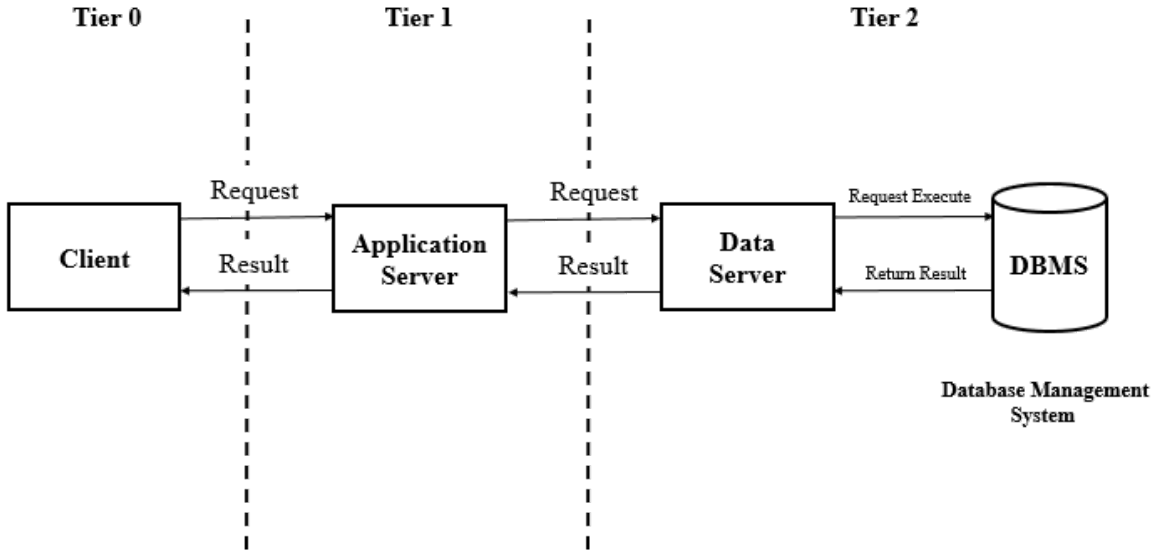
```
class DB_Service {
   connection = DB_Connector.getConnection();

   Account Create (id, pin, name){
      account = INSERT TO ACCOUNT VALUES (id, pin, name, 0)
      return account;
   }
   Account Read (id){
      account = SELECT * FROM ACCOUNT WHERE this.id = id;
      return account;
   }
   Account Update (id, balance) {
      account = UPDATE ACCOUNT SET this.balance = balance WHERE this.id = id;
      return account;
   }

   Account Delete (id){
      DELETE FROM ACCOUNT WHERE this.id = id;
   }
}
```

Πρωτόκολλο Επικοινωνίας

Η ακολουθία των μηνυμάτων θα είναι η εξής:



Τύποι Δεδομένων πρωτοκόλλου επικοινωνίας

- Τύπος αιτήματος RequestType (Αυθεντικοποίηση, κατάθεση, ανάληψη, υπόλοιπο, εγγραφή, αποσύνδεση)

```
enum RequestType{
  auth,
  deposit,
  withdraw,
  checkBalance,
  register,
  logout
}
```

- Αντικείμενο αιτήματος

```
class Request {
    private RequestType type;
    private int id;
    private int pin;
    private double amount;
}
```

- Αντικείμενο απόκρισης

```
class Response {
    private int id;
    private String name;
    private double balance;
    private String message;
    private boolean ok;
}
```

Πρωτόκολλο επικοινωνίας Πελάτη - Διακομιστή

Ο πελάτης μπορεί να εκτελέσει τις εξής ενέργειες

- Εγγραφή
- Σύνδεση/αυθεντικοποίηση
- κατάθεση
- ανάληψη
- Υπόλοιπο
- Αποσύνδεση

Ψευδοκώδικας Αιτημάτων

```
Request register (id, pin, name){
    this.type = Register;
    this.id = id;
    this.pin = pin;
    this.name = name;
}
Request Authentication (id, pin){
    this.type = Auth;
    this.id = id;
    this.pin = pin;
}
Request deposit (amount){
    this.type = Deposit;
    this.amount = amount;
}
Request withdraw (amount){
    this.type = Withdraw;
    this.amount = amount
}
Request balance (){
    this.type = balance;
}
Request logout (){
    this.type = Logout;
}
```

Ψευδοκώδικας Αποκρίσεων

```
Response Success (){
    this.ok = true
}
Response Failed (){
   this.ok = false
   // Εξειδικευμένα μηνύματα ανάλογα το λάθος
   this.message = "Error occurred"
}
Response Auth (){
    this.ok = true;
    // Επιστρέφουμε το όνομα του χρήστη ώστε να το δείξει στην οθόνη
    this.name = name
}
Response Balance (){
    this.ok = true;
    this.balance = balance; }
```

Πρωτόκολλο Διακομιστή

Θέλουμε ο διακομιστής να μπορεί να εξυπηρετήσει πολλαπλούς πελάτες, για αυτόν τον λόγο θα πρέπει να σχεδιάσουμε έναν πολυνηματικό διακομιστή ο οποίος θα δέχεται συνδέσεις πελατών θα δημιουργεί κατάλληλο νήμα για να τις διαχειριστεί. Το νήμα θα είναι υπεύθυνο για την σύνδεση με την βάση, την ανταλλαγή μηνυμάτων αλλα και την επιχειρηματική λογική του συστήματος

Ψευδοκώδικας Πολυνηματικού Διακομιστή

```
class Server{
   create ServerSocket(PORT);
   while(true){
      clientSocket = ServerSocket.listen();
      create ControllerThread(clientSocket);
      controllerThread.start();
   }
}
```

Ψευδοκώδικας για ControllerThread

```
class ControllerThread extends Thread{
   // Constructor
   ControllerThread(clientSocket){
      this.clientSocket = clientSocket;
      this.in = clientSocket.getInputStream();
      this.out = clientSocket.getOutputStream();
      // connection to database for each thread
      service = new DBService();
   }
   // Runs when thread is started
   void run(){
      while (true){
         // get request
         Request request = in.getRequest();
         // create server protocol object
         ServerProtocol serverProtocol = new ServerProtol(request, service);
         // wait for response from server protocol
         Response response = serverProtocol.processRequest();
         // write response to client
         out.writeResponse(response);
      }
   }
}
```

Ψευδοκώδικας για ServerProtocol (Επιχειρηματική λογική)

Στο κομμάτι αυτό γίνεται και διαχείριση σφαλμάτων αλλα και διαχείριση προβλημάτων ταυτοχρονισμού

```
class ServerProtocol{
   ServerProtocol(request, service){
      processRequest(request);
   }
    public Response processRequest(request) {
      switch (request.getType()) {
         case auth:
            return processAuth();
         case register:
            return createAcc();
         case deposit:
            return processDeposit();
         case withdraw:
            return processWithdraw();
         case checkBalance:
            return processCheckBalance();
         case logout:
            return logout();
      }
   }



   private Response logout() {
      Response response = Response.createGeneralSuccessResponse();
      return response;
   }




   private Response processAuth() {

      Account acc = service.auth(id, pin);

      if (acc == null) {
         return Response Error("Authentication failed");
      }

      return Response AuthSuccess(acc.name)
   }
```

```
private Response createAcc() {
    Account foundAccount = service.read(id);

    if (foundAccount != null) {
        return  Response Error("Account already exists");
    }

    lock(id);

    Account account = new Account(id, pin, name, 0);

    if (service.create(account) == null) {
        return  Response Error("Account creation failed please try again");
    }

    unlock(id);
    return Response Success();
}

private Response processDeposit() {

    lock(id);

    double amountToDeposit = request.getAmount();

    if (amountToDeposit == 0) {
        return Response Error("The amount cannot be 0");
    }
    Account accountFound = service.read(id);

    if (accountFound == null) {
        return Response Error("There was an error with the request");
    }

    accountFound.deposit(amountToDeposit);

    if (service.update(accountFound) == null) {
        return Response Error("Deposit failed");
    }

    unlock(id);
    return Response Success();
}
```

```
    private Response processWithdraw() {
        lock(id);
        double amountToWithdraw = request.getAmount();

        if (amountToWithdraw == 0) {
            return Response Error("The amount cannot be 0");
        }

        Account accountFound = service.read(id);

        if (accountFound == null) {
            return Response Error("There was an error with the request");
        }

        if (accountFound.getBalance() - amountToWithdraw < 0) {
            return Response Error("Balance is not enough Withdrawal failed");
        }

        accountFound.withdraw(amountToWithdraw);

        if (service.update(accountFound) == null) {
            return Response Error("Withdrawal failed");
        }
        unlock(id);
        return Response Success();
    }

    private Response processCheckBalance() {
        lock(id);
        Account account = service.read(id);

        if (account == null) {
            return Response Error("Server error");
        }
        unlock(id);
        return Response Balance(balance);

    }
}
```

Εντοπισμός Σφαλμάτων

Ο εντοπισμός σφαλμάτων αφορά κυρίως τα δεδομένα που στέλνει ο πελάτης σχετικά με την ενέργεια που θέλει να ολοκληρώσει. Η σωστή δομή των μηνυμάτων επιτυγχάνεται καθώς στέλνουμε αντικείμενα request και response μεταξύ του πελάτη και του διακομιστή, που έχουν μια γνωστή δομή. Αν προκύψει λάθος στην δομή του μηνύματος θα πιαστεί από κάποιο exception της γλώσσας προγραμματισμού.

Εντοπισμος Σφαλμάτων Στον διακομιστή

Όπως φαίνεται παραπάνω στον ψευδοκώδικα γίνεται έλεγχος στον διακομιστή
σέ κάθε ενέργεια που ζητάει να εκτελέση ο χρήστης. Πιο συγκεκριμένα γίνονται έλεγχοι που αφορούν:

- Την αυθεντικοποίηση του χρήστη (ελέγχεται αν το id και το pin είναι έγκυρα και υπάρχουν στην βάση)
- Την Ανάληψη ενός ποσού (ελέγχεται αν επαρκεί το διαθέσιμο υπόλοιπο για να ολοκληρωθεί η διαδικασία)
- Την δημιουργία ενός χρήστη (ελέγχεται αν υπάρχει ήδη χρήστης με το ιδιο id)
- Γίνεται έλεγχος για κάθε απάντηση που παίρνουμε από την βάση (Αν η απάντηση είναι null σημαίνει πως προέκυψε κάποιο σφάλμα)

Εντοπισμος Σφαλμάτων Στον Πελάτη

Μπορούμε να ελέγχουμε για κάποια προφανή λάθη και στο επίπεδο της παρουσίασης ώστε να κάνουμε το σύστημα πιο εύχρηστο και να μην χρειαστεί να στέλνουμε λάθος δεδομένα στον σέρβερ να ελέγξει. Τέτοιο έλεγχοι είναι οι εξής:

- Έλεγχος για το αν υπάρχουν κενές μεταβλητες π χ κενό όνομα
- Έλεγχος για το αν το id και το pin είναι ακέραιοι αριθμοί

Πιθανές καταστάσεις (states) του πρωτοκόλλου

Ένα πρωτόκολλο θεωρείται ότι έχει μια κατάσταση είναι δηλαδή stateful, όταν το κάθε αίτημα εξαρτάται από την προηγούμενη απόκριση. Ενώ ένα πρωτόκολλο είναι stateless όταν κάθε αίτημα είναι ανεξάρτητο από κάθε προηγούμενη απόκριση. Καλό είναι να αποφεύγουμε τις εξαρτήσεις και το stateful πρωτόκολλο καθώς είναι πιο δύσκολο στην ανάπτυξη και πιό επιρρεπή σε λάθη. Στην περίπτωση μας η μοναδική κατάσταση που θα μπορούσαμε να έχουμε είναι η αυθεντικοποίηση του χρήστη, καθώς για να κάνει τις περισσότερες ενέργειες θα πρέπει να είναι συνδεδεμένος

Προβλήματα Ταυτοχρονισμού

Προβλήματα ταυτοχρονισμού μπορούν να προκύψουν αν επιτρέπεται παραπάνω από μία σύνδεση σε κοινό λογαριασμό. Συγκεκριμένα όταν 2 χρήστες να προσπαθούν να πραγματοποιήσουν ανάληψη στον ίδιο λογαριασμό και δεν μην επαρκεί το ποσό. Στην σχεδίαση μας διαχειριζόμαστε τέτοιες καταστάσεις στο επίπεδο της επιχειρηματικής λογικής με το να κλειδώνουμε το id του λογαριασμού που ξεκινά μια συναλλαγή και το απελευθερώνουμε μόλις ολοκληρωθεί. Επίσης περαιτέρω έλεγχοι περνάνε στο επίπεδο των δεδομένων καθώς σχεδόν κάθε υλοποίηση μιας βάσης δεδομένων έχει λειτουργίες ταυτοχρονισμού. Παραδείγματα κλειδώματος υπάρχουν στον ψευδοκώδικα της επιχειρηματικής λογικής παραπάνω (ServeProtocol)