Cerințe Workout

Cerințe:

1. Implementează funcții pentru încărcarea diferitelor tipuri de date: utilizatori, produse, comenzi, statistici.
2. Folosește `Promise.all()` pentru încărcarea paralelă a datelor independente.
3. Implementează o secvență pentru încărcarea datelor dependente.
4. Adaugă gestionarea erorilor și retry logic pentru operațiunile eșuate.

Cod de pornire: vezi pag 2

GENERAȚIA TECH

```javascript
// Simulare funcții de încărcare date
function loadUsers() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const random = Math.random()
      if (random > 0.2) {
        resolve([
          { id: 1, name: 'Ana Popescu' },
          { id: 2, name: 'Ion Ionescu' },
          { id: 3, name: 'Maria Dumitrescu' },
        ])
      } else {
        reject(new Error('Eroare la încărcarea utilizatorilor'))
      }
    }, 2000)
  })
}

function loadProducts() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const random = Math.random()
      if (random > 0.15) {
        resolve([
          { id: 1, name: 'Laptop', price: 3500 },
          { id: 2, name: 'Telefon', price: 2000 },
          { id: 3, name: 'Tabletă', price: 1200 },
        ])
      } else {
        reject(new Error('Eroare la încărcarea produselor'))
      }
    }, 1500)
  })
}

function loadOrders() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const random = Math.random()
      if (random > 0.1) {
        resolve([
          { id: 1, userId: 1, productId: 1, quantity: 1 },
          { id: 2, userId: 2, productId: 2, quantity: 2 },
          { id: 3, userId: 3, productId: 3, quantity: 1 },
        ])
      } else {
        reject(new Error('Eroare la încărcarea comenzilor'))
      }
    }, 3000)
  })
}

function calculateStatistics(users, products, orders) {
  return new Promise((resolve) => {
    setTimeout(() => {
      const stats = {
        totalUsers: users.length,
        totalProducts: products.length,
        totalOrders: orders.length,
        averageOrderValue:
          products.reduce((sum, p) => sum + p.price, 0) / products.length,
      }
      resolve(stats)
    }, 500)
  })
}

function generateReport(data, statistics) {
  return new Promise((resolve) => {
    setTimeout(() => {
      const report = {
        generatedAt: new Date().toISOString(),
        data,
        statistics,
        summary: `Dashboard conține ${data.users.length} utilizatori, ${data.products.length} produse și
${data.orders.length} comenzi.`,
      }
      resolve(report)
    }, 300)
  })
}

// Funcția principală pentru încărcarea dashboard-ului
function loadDashboard() {
  console.log('🔄 Începe încărcarea dashboard-ului...')

  // Încărcăm datele independente în paralel cu retry logic
  return Promise.all([
    retryOperation(loadUsers),
    retryOperation(loadProducts),
    retryOperation(loadOrders),
  ]).then(([users, products, orders]) => {
    console.log('✅ Date de bază încărcate cu succes')

    const data = { users, products, orders }

    // Calculăm statisticile pe baza datelor încărcate
    return calculateStatistics(users, products, orders).then((statistics) => {
      console.log('✅ Statistici calculate')
      return generateReport(data, statistics)
    })
  })
}

// Funcția de retry pentru operațiuni eșuate
function retryOperation(operation, maxRetries = 3) {
  return new Promise((resolve, reject) => {
    let attempt = 0

    function tryOperation() {
      attempt++
      operation()
        .then(resolve)
        .catch((error) => {
          if (attempt < maxRetries) {
            console.log(
              `⚠️  Încercarea ${attempt} a eșuat, reîncerc... (${
                maxRetries - attempt
              } încercări rămase)`
            )
            setTimeout(tryOperation, 1000 * attempt) // Backoff progresiv
          } else {
            reject(
              new Error(
                `Operațiunea a eșuat după ${maxRetries} încercări: ${error.message}`
              )
            )
          }
        })
    }

    tryOperation()
  })
}

// Testare
loadDashboard()
  .then((dashboard) => {
    console.log('🎉 Dashboard încărcat cu succes!')
    console.log(dashboard)
  })
  .catch((error) => {
    console.error('❌ Eroare la încărcarea dashboard-ului:', error.message)
  })
```