Soluții pentru Exercițiile din Modulul 7

Lecția 2: NPM și Managementul Pachetelor

Colorful lodash demo

```
# 1. Creează proiectul mkdir lodash-demo cd lodash-demo npm init -y # 2. Inst
alează pachetele npm install lodash chalk
```

```
// lodash-demo.js - Demonstrație funcții lodash cu output colorat const _ = r
equire('lodash') const chalk = require('chalk') // Date pentru demonstrație c
onst numbers = [10, 25, 30, 15, 40, 5, 20] const prices = [19.99, 5.5, 12.75,
8.25, 45.0] console.log(chalk.blue.bold('🚀 Demo Lodash Functions cu NPM
\\n')) // 1. Funcții matematice lodash console.log(chalk.green('📊 Lodash Mat
h Functions:')) console.log(`Numbers: ${chalk.white(numbers.join(', '))}`) co
nsole.log(`_.sum(): ${chalk.yellow(_.sum(numbers))}`) console.log(`_.mean():
${chalk.yellow(_.mean(numbers).toFixed(2))}`) console.log(`_.max(): ${chalk.y
ellow(_.max(numbers))}`) console.log(`_.min(): ${chalk.yellow(_.min(number
s))}\\n`) // 2. Funcții pentru array-uri lodash console.log(chalk.green('🔄 L
odash Array Functions:')) const doubled = _.map(numbers, (n) => n * 2) const
evenNumbers = _.filter(numbers, (n) => n % 2 === 0) const sorted = _.sortBy(n
umbers) console.log(`_.map() - doubled: ${chalk.white(doubled.join(', '))}`)
console.log(`_.filter() - even numbers: ${chalk.white(evenNumbers.join(',
'))}`) console.log(`_.sortBy() - sorted: ${chalk.white(sorted.join(', '))}\\n
`) // 3. Demo cu numere (prețuri) console.log(chalk.green('💰 Lodash Number P
rocessing:')) console.log(`Prices: ${chalk.white(prices.map((p) => `$${p}`).j
oin(', '))}`) const totalPrice = _.sum(prices) const averagePrice = _.mean(pr
ices) const expensiveItems = _.filter(prices, (price) => price > 15) console.
log(`_.sum() total: ${chalk.red(`$${totalPrice.toFixed(2)}`)}`) console.log(`
_.mean() average: ${chalk.yellow(`$${averagePrice.toFixed(2)}`)}`) console.lo
g( `_.filter() expensive: ${chalk.magenta( expensiveItems.map((p) => `$${p}
`).join(', ') )}\\n` ) // 4. Funcții utility lodash console.log(chalk.green
('🎲 Lodash Utility Functions:')) console.log(`_.random(1,100): ${chalk.cyan
(_.random(1, 100))}`) console.log(`_.sample() from array: ${chalk.cyan(_.samp
le(numbers))}`) console.log( `_.shuffle() array: ${chalk.white(_.shuffle([1,
2, 3, 4, 5]).join(', '))}\\n` ) // 5. Funcții pentru string-uri lodash const
text = 'hello world from lodash demo' console.log(chalk.green('📝 Lodash Stri
ng Functions:')) console.log(`Original: ${chalk.white(text)}`) console.log(`
_.capitalize(): ${chalk.yellow(_.capitalize(text))}`) console.log(`_.upperCas
e(): ${chalk.cyan(_.upperCase(text))}`) console.log(`_.words(): ${chalk.magen
ta(_.words(text).join(' | '))}\\n`) // 6. Funcții pentru obiecte lodash const
students = [ { name: 'Ana', grade: 9.5, subject: 'Math' }, { name: 'Ion', gra
de: 8.0, subject: 'Physics' }, { name: 'Maria', grade: 9.8, subject: 'Chemist
ry' }, { name: 'Alex', grade: 7.5, subject: 'Math' }, ] console.log(chalk.gre
en('🎓 Lodash Object Functions:')) const avgGrade = _.meanBy(students, 'grad
e') const topStudent = _.maxBy(students, 'grade') const mathStudents = _.filt
er(students, { subject: 'Math' }) console.log(`_.meanBy() average: ${chalk.ye
llow(avgGrade.toFixed(2))}`) console.log( `_.maxBy() top student: ${chalk.gre
en(topStudent.name)} (${topStudent.grade})` ) console.log( `_.filter() + _.ma
p(): ${chalk.blue( _.map(mathStudents, 'name').join(', ') )}\\n` ) console.lo
g(chalk.yellow('✨ Demo lodash funcționează perfect cu NPM! ✨'))
```

```
// package.json rezultat { "name": "lodash-demo", "version": "1.0.0", "descri
ption": "Demonstrație funcții lodash cu output colorat", "main": "lodash-dem
o.js", "scripts": { "start": "node lodash-demo.js" }, "dependencies": { "chal
k": "^4.1.2", "lodash": "^4.17.21" }, "keywords": ["lodash", "demo", "npm",
"learning"], "author": "Your Name", "license": "MIT" }
```

```
# Testează demo-ul lodash npm start
```

Lecția 3: Scripturi și Module Node.js

Sistem de Logging Simplu

Structura proiectului:

```
simple-logger/ ├── src/ │ ├── formatter.js │ ├── logger.js │ └── app.js ├── p
ackage.json └── README.md
```

1. Formatter Module (src/formatter.js)

```
// src/formatter.js - Formatarea mesajelor de log const colors = { INFO: '\\x
1b[32m', // verde WARN: '\\x1b[33m', // galben ERROR: '\\x1b[31m', // roșu re
set: '\\x1b[0m', // reset culoare } export const formatMessage = (level, mess
age) => { const timestamp = new Date().toISOString() const color = colors[lev
el] || colors.reset return `${color}[${timestamp}] ${level}: ${message}${colo
rs.reset}` } // Exportă și culorile pentru folosire în alte module export { c
olors }
```

2. Logger Module (src/logger.js)

```
// src/logger.js - Modulul principal de logging import { formatMessage } from
'./formatter.js' // Validează nivelul de log const validateLogLevel = (level)
=> { const validLevels = ['INFO', 'WARN', 'ERROR', 'DEBUG'] if (!validLevels.
includes(level)) { throw new Error( `Invalid log level: ${level}. Must be one
of: ${validLevels.join(', ')}` ) } } // Funcția principală de logging export
const log = (level, message) => { try { // Validare input if (!message || typ
eof message !== 'string') { throw new Error('Message must be a non-empty stri
ng') } validateLogLevel(level) // Formatează și afișează mesajul const format
tedMessage = formatMessage(level, message) console.log(formattedMessage) } ca
tch (error) { // Gestionare eroare în logger (nu vrem să oprească aplicația)
console.error(`Logger Error: ${error.message}`) } } // Funcții convenience pe
ntru nivele specifice export const info = (message) => log('INFO', message) e
xport const warn = (message) => log('WARN', message) export const error = (me
ssage) => log('ERROR', message) export const debug = (message) => log('DEBU
G', message)
```

3. Test Application (src/app.js)

```
// src/app.js - Testarea sistemului de logging import { log, info, warn, erro
r, debug } from './logger.js' console.log('🚀 Testing Simple Logger System
\\n') // Test funcțiile convenience info('Aplicația a pornit cu succes') warn
('Atenție: memorie puțină disponibilă') error('Eroare în conectarea la baza d
e date') debug('Debug info: processing user request') console.log('\\n📋 Test
ing direct log function:') // Test funcția log direct log('INFO', 'Mesaj info
rmativ direct') log('WARN', 'Mesaj de avertisment direct') log('ERROR', 'Mesa
j de eroare direct') console.log('\\n⚠️ Testing error handling:') // Test ges
tionarea erorilor try { log('INVALID', 'Test invalid level') } catch (error)
{ console.log('Eroarea a fost gestionată corect') } try { log('INFO', '') //
mesaj gol } catch (error) { console.log('Mesajul gol a fost gestionat corec
t') } try { log('INFO', null) // mesaj null } catch (error) { console.log('Me
sajul null a fost gestionat corect') } console.log('\\n✅ Logger system testi
ng complete!')
```

4. Package Configuration (package.json)

```
{ "name": "simple-logger", "version": "1.0.0", "description": "Sistem simplu de logging pentru Node.js cu module și formatare colorată", "type": "module", "main": "src/app.js", "scripts": { "start": "node src/app.js", "test": "node src/app.js" }, "keywords": ["logging", "nodejs", "modules", "colors"], "author": "Your Name", "license": "MIT" }
```

Output așteptat:

```
🚀 Testing Simple Logger System [2024-01-15T10:30:00.123Z] INFO: Aplicația a pornit cu succes [2024-01-15T10:30:00.124Z] WARN: Atenție: memorie puțină disponibilă [2024-01-15T10:30:00.125Z] ERROR: Eroare în conectarea la baza de date [2024-01-15T10:30:00.126Z] DEBUG: Debug info: processing user request 📋 Testing direct log function: [2024-01-15T10:30:00.127Z] INFO: Mesaj informativ direct [2024-01-15T10:30:00.128Z] WARN: Mesaj de avertisment direct [2024-01-15T10:30:00.129Z] ERROR: Mesaj de eroare direct ⚠️ Testing error handling: Logger Error: Invalid log level: INVALID. Must be one of: INFO, WARN, ERROR, DEBUG Logger Error: Message must be a non-empty string Logger Error: Message must be a non-empty string ✅ Logger system testing complete!
```

Lecția 4: Lucrul cu Fișiere și Mediul de Execuție

Extinderea Logger-ului Avansat

1. Configuration Module

```
// src/config.js - Configurarea logger-ului prin environment variables import
path from 'path' const config = { logLevel: process.env.LOG_LEVEL || 'INFO',
logToFile: process.env.LOG_TO_FILE === 'true', logDir: process.env.LOG_DIR ||
'./logs', maxLogSize: parseInt(process.env.MAX_LOG_SIZE) || 1024 * 1024, // 1
MB default // Niveluri de log cu priorități levels: { DEBUG: 0, INFO: 1, WAR
N: 2, ERROR: 3, }, // Validarea configurației validate() { if (!this.levels[t
his.logLevel]) { throw new Error( `Invalid LOG_LEVEL: ${this.logLevel}. Must
be one of: ${Object.keys( this.levels ).join(', ')}` ) } if (this.maxLogSize
< 1024) { throw new Error('MAX_LOG_SIZE must be at least 1024 bytes') } retur
n true }, // Verifică dacă un nivel de log ar trebui să fie afișat shouldLog
(level) { const currentLevel = this.levels[this.logLevel] const messageLevel
= this.levels[level] return messageLevel >= currentLevel }, } export default
config
```

2. Formatter Module

```
// src/formatter.js - Formatarea mesajelor const colors = { DEBUG: '\\x1b[36
m', // cyan INFO: '\\x1b[32m', // verde WARN: '\\x1b[33m', // galben ERROR:
'\\x1b[31m', // roșu reset: '\\x1b[0m', } export const formatForConsole = (le
vel, message, meta = {}) => { const timestamp = new Date().toISOString() cons
t color = colors[level] || colors.reset const metaStr = Object.keys(meta).len
gth > 0 ? ` ${JSON.stringify(meta)}` : '' return `${color}[${timestamp}] ${le
vel}: ${message}${metaStr}${colors.reset}` } export const formatForFile = (le
vel, message, meta = {}) => { const timestamp = new Date().toISOString() cons
t logEntry = { timestamp, level, message, ...meta, } return JSON.stringify(lo
gEntry) + '\\n' } export { colors }
```

3. File Writer Module

```
// src/file-writer.js - Gestionarea fișierelor de log import fs from 'fs' imp
ort path from 'path' import { promises as fsPromises } from 'fs' import confi
g from './config.js' class FileWriter { constructor() { this.streams = new Ma
p() this.isShuttingDown = false // Graceful shutdown handlers process.on('SIG
INT', () => this.shutdown()) process.on('SIGTERM', () => this.shutdown()) } a
sync ensureLogDirectory() { try { await fsPromises.mkdir(config.logDir, { rec
ursive: true }) } catch (error) { console.error('Failed to create log directo
ry:', error.message) throw error } } async checkFileRotation(filePath) { try
{ const stats = await fsPromises.stat(filePath) if (stats.size > config.maxLo
gSize) { const timestamp = new Date().toISOString().replace(/[:.]/g, '-') con
st rotatedPath = `${filePath}.${timestamp}` await fsPromises.rename(filePath,
rotatedPath) console.log( `Log file rotated: ${path.basename(filePath)} ->
${path.basename( rotatedPath )}` ) return true } } catch (error) { if (error.
code !== 'ENOENT') { console.error('Error checking file rotation:', error.mes
sage) } } return false } getStream(filename) { if (!this.streams.has(filenam
e)) { const filePath = path.join(config.logDir, filename) const stream = fs.c
reateWriteStream(filePath, { flags: 'a' }) stream.on('error', (error) => { co
nsole.error(`Log file write error for ${filename}:`, error.message) }) this.s
treams.set(filename, stream) } return this.streams.get(filename) } async writ
e(level, formattedMessage) { if (this.isShuttingDown) return try { await thi
s.ensureLogDirectory() // Scrie în fișierul specific nivelului const levelFil
e = `${level.toLowerCase()}.log` await this.checkFileRotation(path.join(confi
g.logDir, levelFile)) const levelStream = this.getStream(levelFile) levelStre
am.write(formattedMessage) // Scrie și în fișierul general const allFile = 'a
ll.log' await this.checkFileRotation(path.join(config.logDir, allFile)) const
allStream = this.getStream(allFile) allStream.write(formattedMessage) } catch
(error) { console.error('Failed to write to log file:', error.message) } } as
ync shutdown() { this.isShuttingDown = true const closePromises = Array.from
(this.streams.values()).map((stream) => { return new Promise((resolve) => { s
tream.end(resolve) }) }) await Promise.all(closePromises) this.streams.clear
() console.log('File writer shutdown complete') } } export default new FileWr
iter()
```

4. Core Logger Module

```
// src/core.js - Logger principal integrat (evoluat din logger.js) import con
fig from './config.js' import { formatForConsole, formatForFile } from './for
matter.js' import fileWriter from './file-writer.js' class Logger { construct
or() { // Validează configurația la inițializare try { config.validate() cons
ole.log( `Logger initialized - Level: ${config.logLevel}, File: ${config.logT
oFile}` ) } catch (error) { console.error('Logger configuration error:', erro
r.message) throw error } } log(level, message, meta = {}) { try { // Verifică
dacă nivelul ar trebui să fie afișat if (!config.shouldLog(level)) { return }
// Validare input if (!message || typeof message !== 'string') { throw new Er
ror('Message must be a non-empty string') } // Formatează pentru consolă cons
t consoleMessage = formatForConsole(level, message, meta) console.log(console
Message) // Scrie în fișier dacă este activat if (config.logToFile) { const f
ileMessage = formatForFile(level, message, meta) fileWriter.write(level, file
Message).catch((error) => { console.error('Logger file write failed:', error.
message) }) } } catch (error) { console.error(`Logger error: ${error.message}
`) } } debug(message, meta = {}) { this.log('DEBUG', message, meta) } info(me
ssage, meta = {}) { this.log('INFO', message, meta) } warn(message, meta =
{}) { this.log('WARN', message, meta) } error(message, error = null, meta =
{}) { const errorMeta = error ? { error: error.message, stack: error.stack,
...meta, } : meta this.log('ERROR', message, errorMeta) } async shutdown() {
console.log('Logger shutting down...') await fileWriter.shutdown() } } export
default new Logger()
```