Node.js e Express: Guida Completa

Fondamenti di Node.js

Moduli e Sistema di File

```javascript
// Importare moduli core
const fs = require('fs');
const path = require('path');
const os = require('os');

// ES Modules (Node.js moderno)
import fs from 'fs';
import path from 'path';

// Leggere file
fs.readFile('file.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Errore:', err);
    return;
  }
  console.log(data);
});

// Versione promisificata
import { promises as fsPromises } from 'fs';

async function readFileContent() {
  try {
    const data = await fsPromises.readFile('file.txt', 'utf8');
    console.log(data);
  } catch (err) {
    console.error('Errore:', err);
  }
}
```

Eventi e Streams

```javascript
// Eventi
const EventEmitter = require('events');

class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();

myEmitter.on('event', (arg) => {
  console.log('evento scattato!', arg);
});

myEmitter.emit('event', 'arg1');

// Streams
const readStream = fs.createReadStream('file.txt');
const writeStream = fs.createWriteStream('output.txt');

readStream.pipe(writeStream);

readStream.on('data', (chunk) => {
  console.log(`Ricevuti ${chunk.length} byte di dati`);
});

readStream.on('end', () => {
  console.log('Lettura completata');
});
```

HTTP e API Server

```javascript
const http = require('http');

const server = http.createServer((req, res) => {
  if (req.url === '/api/users' && req.method === 'GET') {
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({ users: ['John', 'Jane', 'Bob'] }));
  } else {
    res.writeHead(404);
    res.end('Not Found');
  }
});

server.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

Express Framework

Setup di Base

javascript

```javascript
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

// Middleware globali
app.use(express.json()); // Per parsing application/json
app.use(express.urlencoded({ extended: true })); // Per parsing form data

// Middleware per logging
app.use((req, res, next) => {
  console.log(`${req.method} ${req.path}`);
  next();
});

// Route di base
app.get('/', (req, res) => {
  res.send('Hello World!');
});

// Avvio del server
app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

Routing in Express

javascript

```javascript
// Route con parametri
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  // Logica per recuperare l'utente
  res.json({ id: userId, name: 'John Doe' });
});

// Gestione diverse HTTP methods
app.post('/users', (req, res) => {
  const newUser = req.body;
  // Logica per creare utente
  res.status(201).json(newUser);
});

app.put('/users/:id', (req, res) => {
  const userId = req.params.id;
  const updatedUser = req.body;
  // Logica per aggiornare utente
  res.json(updatedUser);
});

app.delete('/users/:id', (req, res) => {
  const userId = req.params.id;
  // Logica per eliminare utente
  res.status(204).end();
});

// Route con query parameters
app.get('/products', (req, res) => {
  const { category, sort } = req.query;
  // Logica per filtrare e ordinare prodotti
  res.json({ products: [], filters: { category, sort } });
});
```

Router Modulare

javascript

```javascript
// userRoutes.js
const express = require('express');
const router = express.Router();

// Middleware specifico per questo router
router.use((req, res, next) => {
  console.log('Time:', Date.now());
  next();
});

router.get('/', (req, res) => {
  res.json({ users: [] });
});

router.post('/', (req, res) => {
  // Crea utente
});

module.exports = router;

// app.js
const userRoutes = require('./routes/userRoutes');
app.use('/api/users', userRoutes);
```

Middleware

javascript

```javascript
// Middleware di autenticazione
function authMiddleware(req, res, next) {
  const token = req.headers.authorization;

  if (!token) {
    return res.status(401).json({ message: 'Authentication required' });
  }

  try {
    // Verifica token (esempio)
    const decoded = verifyToken(token);
    req.user = decoded;
    next();
  } catch (error) {
    res.status(401).json({ message: 'Invalid token' });
  }
}

// Utilizzo middleware su route specifiche
app.get('/profile', authMiddleware, (req, res) => {
  res.json({ user: req.user });
});

// Middleware di gestione errori
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: 'Something went wrong!' });
});
```

Express con Async/Await

```javascript
// Wrapper per gestire errori nelle funzioni async
function asyncHandler(fn) {
  return (req, res, next) => {
    Promise.resolve(fn(req, res, next)).catch(next);
  };
}

// Utilizzo con route async
app.get('/users', asyncHandler(async (req, res) => {
  const users = await User.findAll();
  res.json(users);
}));

// In alternativa, con Express 5+ o express-async-errors
require('express-async-errors');

app.get('/users', async (req, res) => {
  const users = await User.findAll();
  res.json(users);
});
```

Design Pattern per Node.js

MVC Pattern

```
project/
├── controllers/
│   ├── userController.js
│   └── productController.js
├── models/
│   ├── User.js
│   └── Product.js
├── views/ (per applicazioni SSR)
│   ├── users/
│   └── products/
├── routes/
│   ├── userRoutes.js
│   └── productRoutes.js
├── middleware/
│   ├── auth.js
│   └── error.js
├── services/
│   ├── userService.js
│   └── productService.js
├── config/
│   └── db.js
└── app.js
```

Esempio di controller:

javascript

```javascript
// userController.js
const User = require('../models/User');
const userService = require('../services/userService');

exports.getAllUsers = async (req, res) => {
  try {
    const users = await userService.findAll();
    res.json(users);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

exports.getUserById = async (req, res) => {
  try {
    const user = await userService.findById(req.params.id);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json(user);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

// Implementazioni per create, update, delete...
```

Repository Pattern

javascript

```javascript
// userRepository.js
const User = require('../models/User');

class UserRepository {
  async findAll() {
    return User.find({});
  }

  async findById(id) {
    return User.findById(id);
  }

  async create(userData) {
    const user = new User(userData);
    return user.save();
  }
```