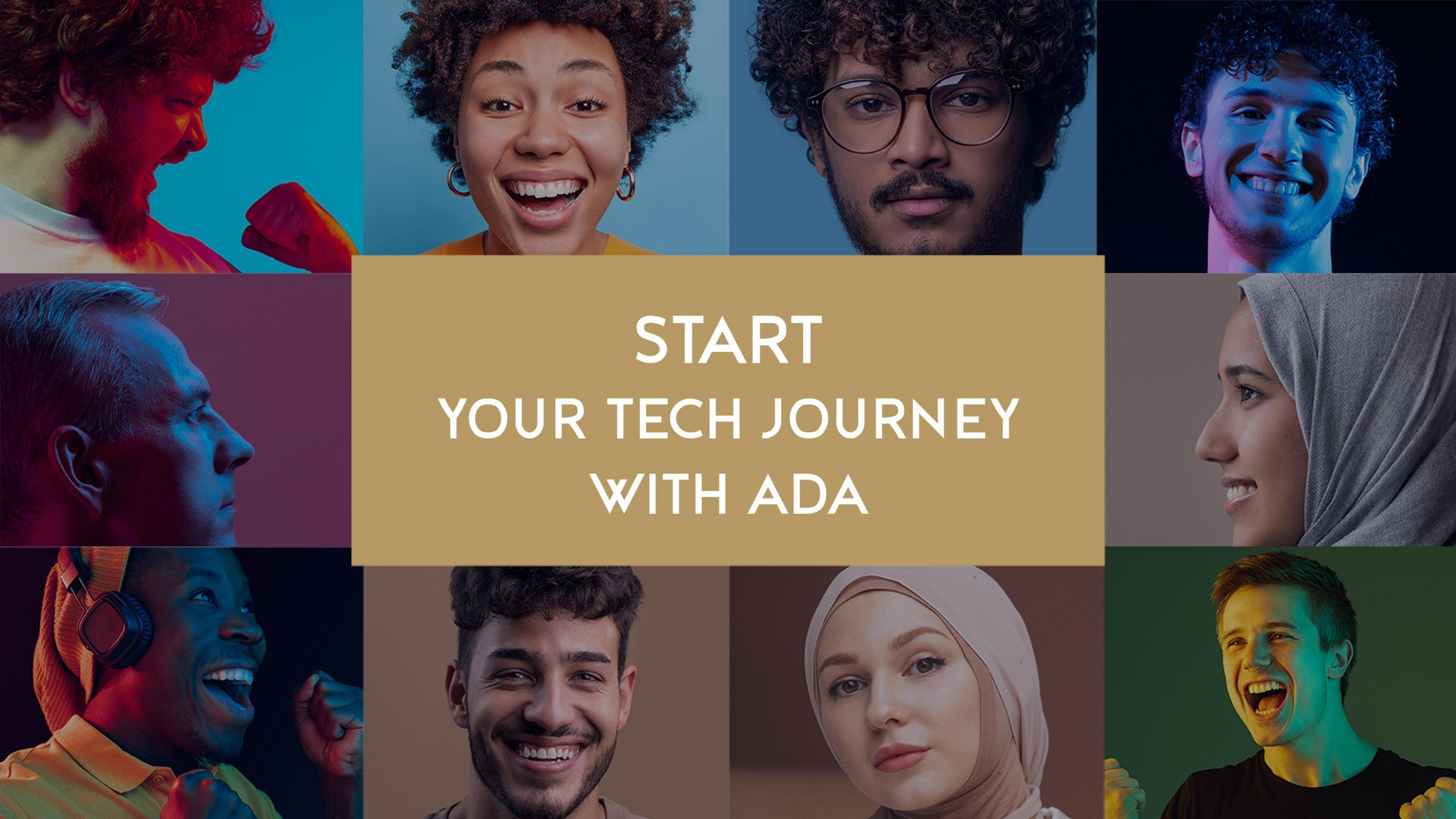
ACADEMY OF DIGITAL ARTS EGYPT

Google cisco Adobe IBM Microsoft CompTIA



START YOUR TECH JOURNEY WITH ADA

Agenda

- ES6+ Features in Node.js: import/export (ES Modules)
- Default vs named exports: Modern module patterns
- Template literals, destructuring, spread/rest: Quick recap in Node context
- Promises in Node.js: Creating and consuming promises
- .then() and .catch(): Promise chaining and error handling
- Async/Await: Writing cleaner asynchronous code
- Error handling with try/catch in async functions
- Converting callback-based functions to promises

ES6 Modules in Node.js

Enabling ES6 Modules in Node.js:

Method 1: package.json Configuration

```
{
    "name": "my-app",
    "version": "1.0.0",
    "type": "module",
    "scripts": {
        "start": "node app.js"
    }
}
```

Method 2: .mjs File Extension

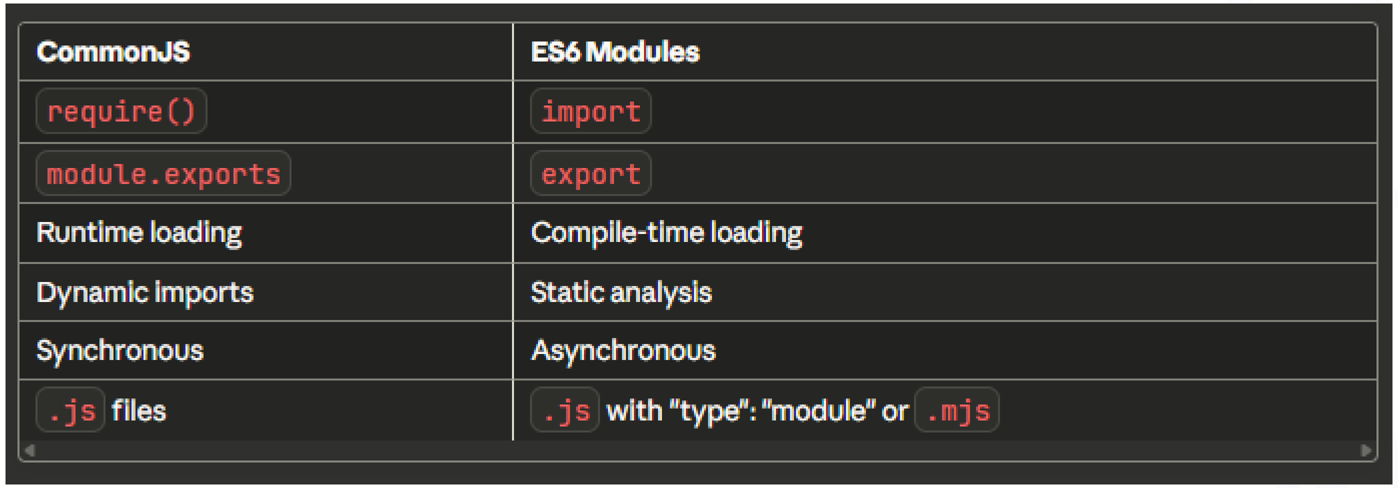
```
// math.mjs - ES6 module file
// app.mjs - Main application file
```

Method 3: Node.js Flag

```
# Run with ES modules flag
node --input-type=module app.js
```

ES6 Modules in Node.js

Comparison: CommonJS vs ES6 Modules



CommonJS	ES6 Modules
`require()`	`import`
`module.exports`	`export`
Runtime loading	Compile-time loading
Dynamic imports	Static analysis
Synchronous	Asynchronous
`.js` files	`.js` with "type": "module" or `.mjs`

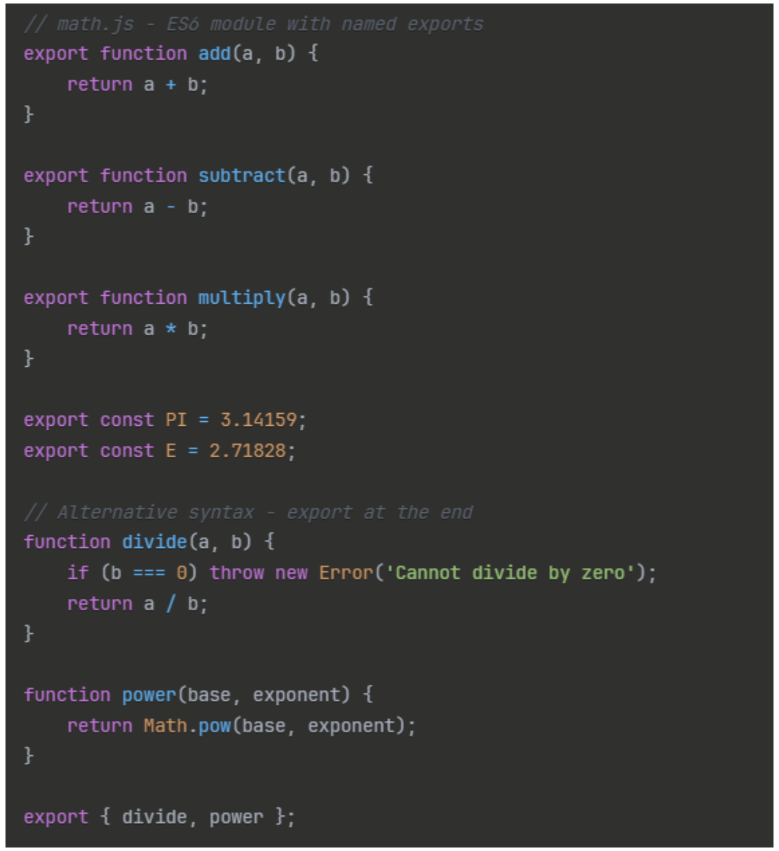
ES6 Modules in Node.js

Why ES6 Modules?

- Static analysis - Better optimization and tree-shaking
- Cleaner syntax - More readable import/export statements
- Standard across platforms - Same syntax in browser and Node.js
- Better IDE support - Improved autocomplete and refactoring
- Future-proof - Modern JavaScript standard

Named Exports and Imports

Named Exports - Multiple Functions:



```javascript
// math.js - ES6 module with named exports
export function add(a, b) {
    return a + b;
}

export function subtract(a, b) {
    return a - b;
}

export function multiply(a, b) {
    return a * b;
}

export const PI = 3.14159;
export const E = 2.71828;

// Alternative syntax - export at the end
function divide(a, b) {
    if (b === 0) throw new Error('Cannot divide by zero');
    return a / b;
}

function power(base, exponent) {
    return Math.pow(base, exponent);
}

export { divide, power };
```

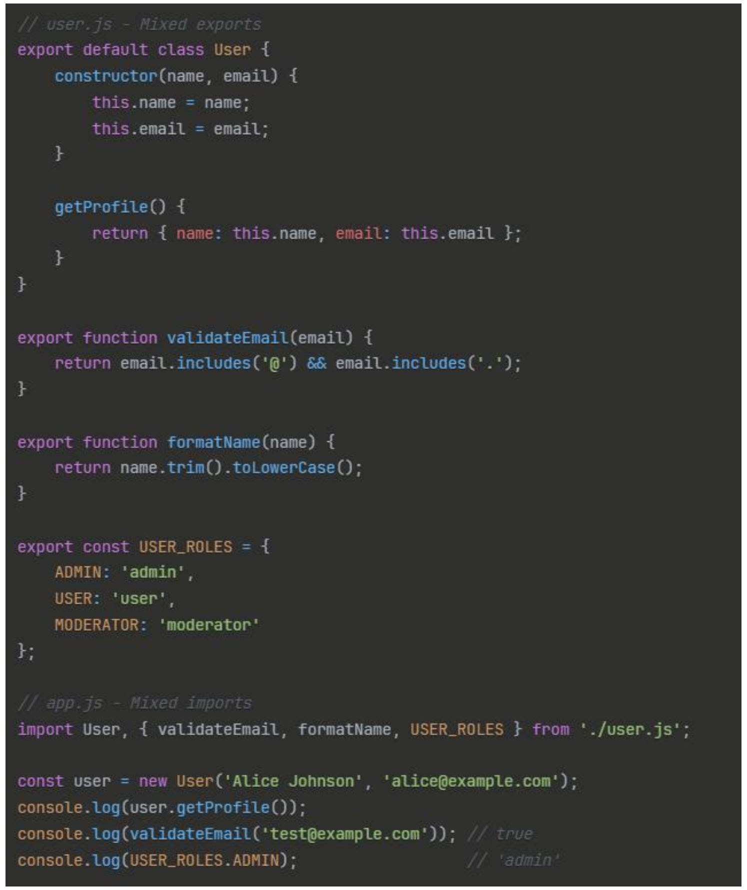
Named Exports and Imports

Named Imports - Selective Importing:

```javascript
// app.js - Importing specific functions
import { add, subtract, PI } from './math.js';

console.log(add(5, 3));         // 8
console.log(subtract(10, 4));   // 6
console.log(`PI = ${PI}`);      // PI = 3.14159

// Import with aliases
import { multiply as mult, divide as div } from './math.js';

console.log(mult(6, 7));        // 42
console.log(div(15, 3));        // 5

// Import everything
import * as MathUtils from './math.js';

console.log(MathUtils.add(2, 3));      // 5
console.log(MathUtils.power(2, 8));    // 256
console.log(MathUtils.PI);             // 3.14159
```

Named Exports and Imports

Mixed Named and Default Exports:



```javascript
// user.js - Mixed exports
export default class User {
    constructor(name, email) {
        this.name = name;
        this.email = email;
    }

    getProfile() {
        return { name: this.name, email: this.email };
    }
}

export function validateEmail(email) {
    return email.includes('@') && email.includes('.');
}

export function formatName(name) {
    return name.trim().toLowerCase();
}

export const USER_ROLES = {
    ADMIN: 'admin',
    USER: 'user',
    MODERATOR: 'moderator'
};

// app.js - Mixed imports
import User, { validateEmail, formatName, USER_ROLES } from './user.js';

const user = new User('Alice Johnson', 'alice@example.com');
console.log(user.getProfile());
console.log(validateEmail('test@example.com')); // true
console.log(USER_ROLES.ADMIN);                  // 'admin'
```

Default Exports and Imports

Default Export Patterns:

```javascript
// logger.js - Default export with class
export default class Logger {
    constructor(name) {
        this.name = name;
    }

    log(message) {
        const timestamp = new Date().toISOString();
        console.log(`[${timestamp}] ${this.name}: ${message}`);
    }

    error(message) {
        const timestamp = new Date().toISOString();
        console.error(`[${timestamp}] ${this.name} ERROR: ${message}`);
    }
}
```

```javascript
// config.js - Default export with object
const config = {
    port: process.env.PORT || 3000,
    database: {
        host: process.env.DB_HOST || 'localhost',
        port: process.env.DB_PORT || 5432
    },
    api: {
        timeout: 5000,
        retries: 3
    }
};

export default config;
```

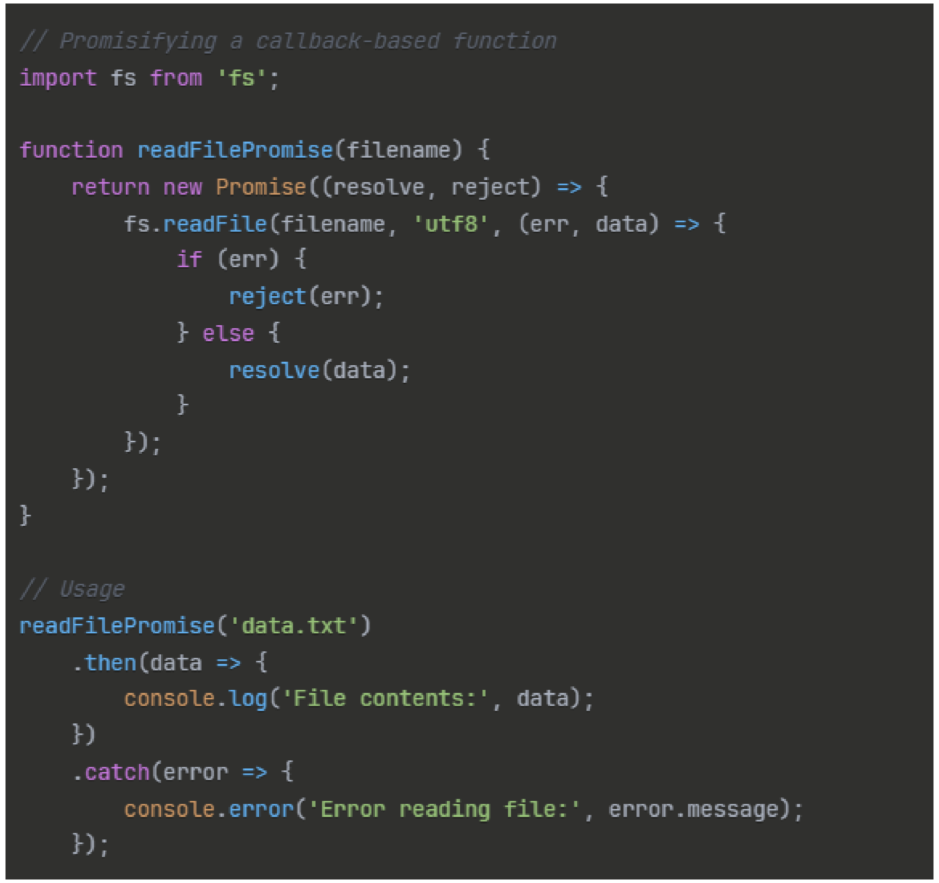
Default Exports and Imports

Default Import Patterns:

```javascript
// app.js - Importing default exports
import Logger from './logger.js';           // Class import
import config from './config.js';           // Object import
import createId from './utils.js';          // Function import

// You can name default imports whatever you want
import MyLogger from './logger.js';         // Same as Logger
import AppConfig from './config.js';        // Same as config
import generateId from './utils.js';        // Same as createId

// Usage
const logger = new Logger('App');
logger.log('Application starting...');

console.log(`Server will run on port ${config.port}`);

const userId = createId();
console.log(`Generated ID: ${userId}`);
```

Promises in Node.js Deep Dive

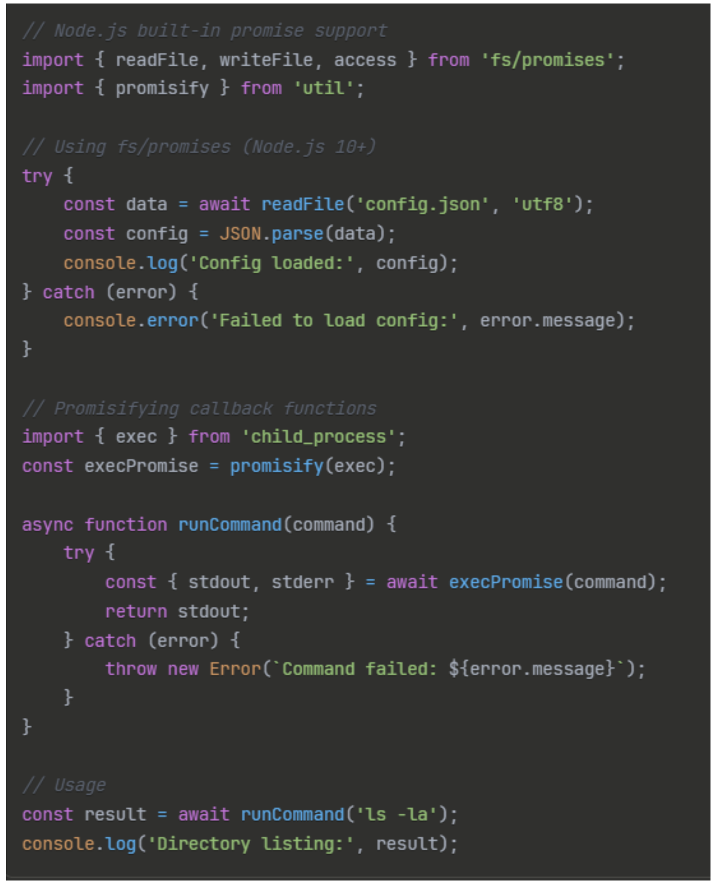
Creating Custom Promises:



```javascript
// Promisifying a callback-based function
import fs from 'fs';

function readFilePromise(filename) {
    return new Promise((resolve, reject) => {
        fs.readFile(filename, 'utf8', (err, data) => {
            if (err) {
                reject(err);
            } else {
                resolve(data);
            }
        });
    });
}

// Usage
readFilePromise('data.txt')
    .then(data => {
        console.log('File contents:', data);
    })
    .catch(error => {
        console.error('Error reading file:', error.message);
    });
```

Promises in Node.js Deep Dive

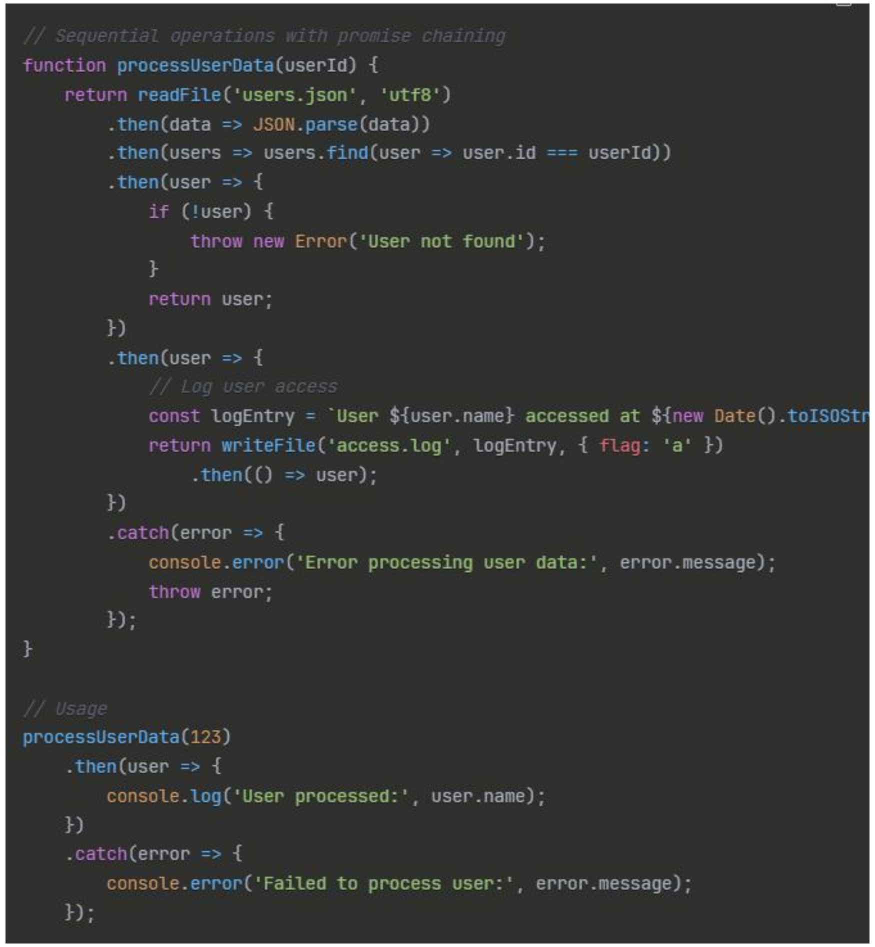
Built-in Promise Support:



```javascript
// Node.js built-in promise support
import { readFile, writeFile, access } from 'fs/promises';
import { promisify } from 'util';

// Using fs/promises (Node.js 10+)
try {
    const data = await readFile('config.json', 'utf8');
    const config = JSON.parse(data);
    console.log('Config loaded:', config);
} catch (error) {
    console.error('Failed to load config:', error.message);
}

// Promisifying callback functions
import { exec } from 'child_process';
const execPromise = promisify(exec);

async function runCommand(command) {
    try {
        const { stdout, stderr } = await execPromise(command);
        return stdout;
    } catch (error) {
        throw new Error(`Command failed: ${error.message}`);
    }
}

// Usage
const result = await runCommand('ls -la');
console.log('Directory listing:', result);
```

Promises in Node.js Deep Dive

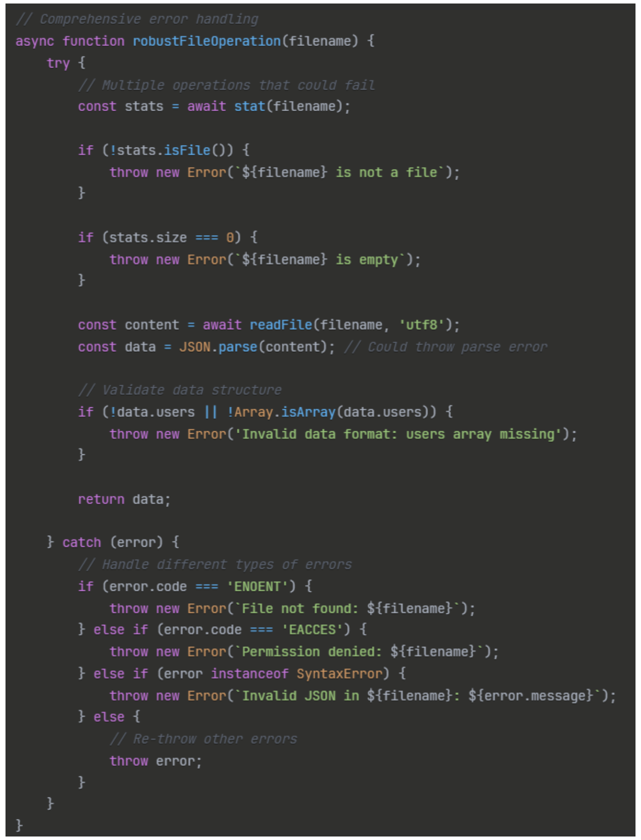
Promise Chaining Patterns:



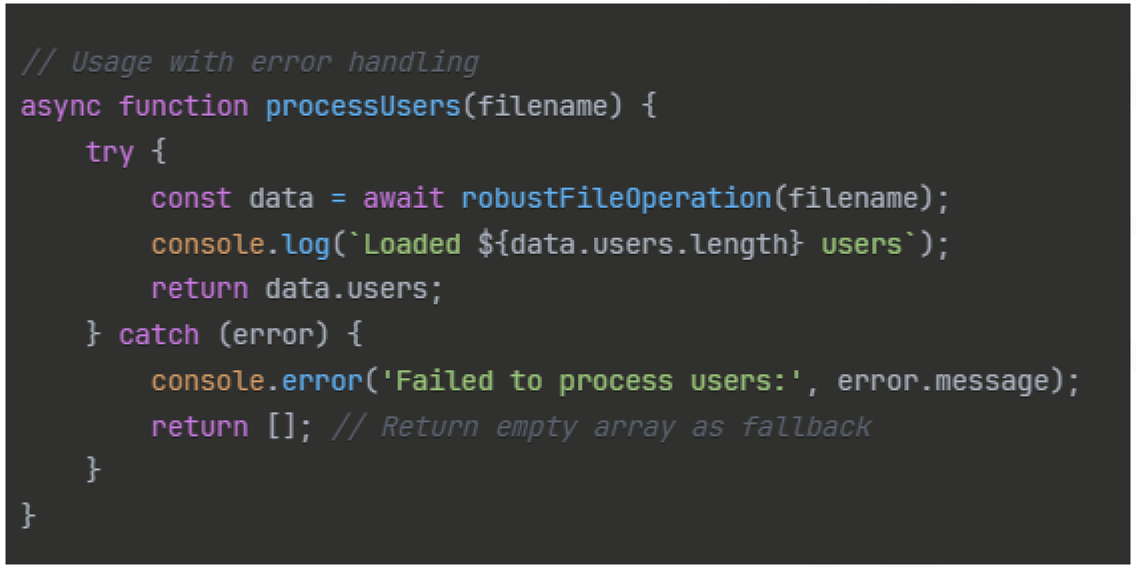
```javascript
// Sequential operations with promise chaining
function processUserData(userId) {
    return readFile('users.json', 'utf8')
        .then(data => JSON.parse(data))
        .then(users => users.find(user => user.id === userId))
        .then(user => {
            if (!user) {
                throw new Error('User not found');
            }
            return user;
        })
        .then(user => {
            // Log user access
            const logEntry = `User ${user.name} accessed at ${new Date().toISOStr
            return writeFile('access.log', logEntry, { flag: 'a' })
                .then(() => user);
        })
        .catch(error => {
            console.error('Error processing user data:', error.message);
            throw error;
        });
}

// Usage
processUserData(123)
    .then(user => {
        console.log('User processed:', user.name);
    })
    .catch(error => {
        console.error('Failed to process user:', error.message);
    });
```

Error Handling with Async/Await

Try/Catch with Async/Await:



```javascript
// Comprehensive error handling
async function robustFileOperation(filename) {
    try {
        // Multiple operations that could fail
        const stats = await stat(filename);

        if (!stats.isFile()) {
            throw new Error(`${filename} is not a file`);
        }

        if (stats.size === 0) {
            throw new Error(`${filename} is empty`);
        }

        const content = await readFile(filename, 'utf8');
        const data = JSON.parse(content); // Could throw parse error

        // Validate data structure
        if (!data.users || !Array.isArray(data.users)) {
            throw new Error('Invalid data format: users array missing');
        }

        return data;

    } catch (error) {
        // Handle different types of errors
        if (error.code === 'ENOENT') {
            throw new Error(`File not found: ${filename}`);
        } else if (error.code === 'EACCES') {
            throw new Error(`Permission denied: ${filename}`);
        } else if (error instanceof SyntaxError) {
            throw new Error(`Invalid JSON in ${filename}: ${error.message}`);
        } else {
            // Re-throw other errors
            throw error;
        }
    }
}
```



```javascript
// Usage with error handling
async function processUsers(filename) {
    try {
        const data = await robustFileOperation(filename);
        console.log(`Loaded ${data.users.length} users`);
        return data.users;
    } catch (error) {
        console.error('Failed to process users:', error.message);
        return []; // Return empty array as fallback
    }
}
```

THANK YOU

ADAEGY