# Intermediate Node.js Tutorial: Modularizing File System, Event Emitter, and Crypto Modules

## Introduction

In this intermediate Node.js tutorial, we will modularize the File System (fs), Event Emitter, and Crypto modules. We'll create separate modules for each functionality and import them into our app.js file.

## Prerequisites

Before we begin, ensure you have Node.js installed on your machine. You can download it from the Node.js website. Additionally, you'll need a code editor of your choice.

### Step 1: Setting up the project

1. Create a new directory for your project and navigate to it in your terminal.
2. Initialize a new Node.js project by running:

```
npm init -y
```

3. Install **nodemon** globally to help us with automatic code reloading:

```
npm install -g nodemon
```

4.Create the following directory structure in your project:

```
├── modules
│   ├── eventEmitter.js
│   ├── fileSystem.js
│   └── crypto.js
└── app.js
```

5.Create a JavaScript file for each module in the modules directory and an app.js file in the root of your project.

## Step 2. Modularizing the File System Module (fileSystem.js)

In the fileSystem.js module, we'll encapsulate the functionality related to the File System (fs) module.

```
// modules/fileSystem.js
const fs = require('fs');

function writeToFile(filename, content) {
  fs.writeFileSync(filename, content);
}

function readFromFile(filename) {
  return fs.readFileSync(filename, 'utf-8');
}

module.exports = {
  writeToFile,
  readFromFile,
};
```

## Step 3. Modularizing the Event Emitter Module (eventEmitter.js)

In the eventEmitter.js module, we'll encapsulate the functionality related to the Event Emitter module.

```
// modules/eventEmitter.js
const EventEmitter = require('events');

const myEmitter = new EventEmitter();

function listenForCustomEvent(callback) {
  myEmitter.on('customEvent', callback);
}

function emitCustomEvent(message) {
  myEmitter.emit('customEvent', message);
}

module.exports = {
  listenForCustomEvent,
  emitCustomEvent,
};
```

## Step 4. Modularizing the Crypto Module (crypto.js)

In the crypto.js module, we'll encapsulate the functionality related to the Crypto module.

```
// modules/crypto.js
const crypto = require('crypto');

function hashPassword(password) {
  const hash = crypto.createHash('sha256').update(password).digest('hex');
  return hash;
}

module.exports = {
  hashPassword,
};
```

## Step 5. Using the Modules in app.js

Now, in your app.js file, you can import and use these modules:

```
// Import the modules
const fsModule = require('./modules/fileSystem');
const eventEmitterModule = require('./modules/eventEmitter');
const cryptoModule = require('./modules/crypto');

// Use the File System module
fsModule.writeToFile('example.txt', 'Hello, File System!');
const content = fsModule.readFromFile('example.txt');
console.log(`File content: ${content}`);

// Use the Event Emitter module
eventEmitterModule.listenForCustomEvent((message) => {
  console.log(`Received custom event: ${message}`);
});
eventEmitterModule.emitCustomEvent('Hello, Node.js!');

// Use the Crypto module
const hashedPassword = cryptoModule.hashPassword('MySecurePassword');
console.log(`Hashed password: ${hashedPassword}`);
```

Now, you have modularized your code, making it more organized and easier to maintain. Save the code in your **app.js** file and run it using **npm** start to see the results.

## Conclusion

By modularizing the File System, Event Emitter, and Crypto modules, you have created a well-structured Node.js application that is easier to manage and maintain. Modularization is a best practice in Node.js development, and it allows you to reuse code efficiently.