Módulos en Node.js

Trabajar con módulos en Node es una muy buena práctica definida por CommonJS ([**http://www.commonjs.org/**](http://www.commonjs.org/)), ya que logras que tus aplicaciones sean escalables y tu código se verá mucho más limpio y organizado.

Se recomienda agrupar las operaciones en módulos de acuerdo a sus funcionalidades, es decir ,tener un módulo para los controladores, uno para las conexiones, uno para los modelos, uno para los middlewares, etc.

El manejo de módulos es una buena práctica aprendida de Django.

**Consejo 1:**Un server.js no debe tener más de 1000 líneas de código.

**Consejo 2:**En el server.js NO debe ir código diferente a la configuración del server.

Los módulos no son más que un fólder dentro de tu aplicación, en donde están los archivos .js relacionados según su funcionalidad.

Estructura básica de un módulo

**1.**Las librerías que va a requerir tu módulo.
**2.** una variable con el nombre del módulo (nombreTipoDeModulo), esta variable contiene una función que como mínimo siempre va a recibir al server como parámetro. Dentro de esta función se declaran todas las funciones que hace tu módulo.

**Consejo 3:** Recuerda siempre, que las funciones son todo en JavaScript

**3.** Exportar el modelo con module.exports; si no exportas el módulo, no vas a tener forma de requerirlo donde los necesites.
 Ejemplo de un módulo de controladores

**var** \_ = **require**('underscore');
**var** homeController = **function** (server, users) {
    console.log('homeController esta cargado');
  **var** isLoggedIn = **function** (req, res, **next**){
        **if**(req.session.passport.user){
     res.redirect('/app');
     **return**;
   }
   **next**();
};
server.**get**('/', isLoggedIn, **function** (req, res) {
    res.render('home');
});
**module**.exports = homeController;

**Consejo 4:**Siempre recuerda cargar las librerías que vas a usar en tu código al inicio; esto hace que tu código sea mucho más amigable para las otras personas que lo vean y/o les toque trabajar con él.

Para requerir tus módulos en el server.js no son más de 2 línea de código.

**1.**Creas una variable con el nombre de la variable de tu modulo y requieres la ruta del archivo .js donde se encuentra el módulo.

Ej

**var** homeController = **require**('./app/controllers/home');

**Nota:** Para moverse entre los folders de tu aplicación se usa ./ para ir hacia adentro desde el punto donde estas y ../ Para ir hacia atrás → ./ - ../ son los equivalentes de cd en la consola.

**2.** Llamas a la función del módulo, pasándole sus respectivos parámetros.

Ej.

homeController(server, users);

Cuando estas creando el módulo para tus modelos en Node.js, se hace necesario tener claro que base de datos vas a utilizar;  si usamos MongoDB + Node.Js existe una librería que hace de este trabajo una tarea fácil.

Mongoose

[**http://mongoosejs.com/**](http://mongoosejs.com/)

Mongoose es una librería para entornos asíncronos como Node, que sirve como herramienta para el modelado de objetos MongoDB.

Mongoose proporciona una solución sencilla, que se basa en esquemas para modelar los datos de la aplicación e incluye funciones de conversión de tipos, validación, generación de consultas y más.

Usar Mongoose es muy fácil y bonito:

**1.** Instalar mongoose a tu aplicación de Node

$ npm install mongoose –save

**2.** Definir la conexión con la base de datos de MongoDB; para esto se debe requerir mongoose y luego con darle la ruta de conexión a la base de datos.

**Nota:**Es recomendable trabajar tus modelos, dentro de un módulo de Models y en models.js definir la conexión.

Ej.

**var** mongoose = **require**('mongoose');
mongoose.connect('mongodb://localhost/' + 'backendPro');
**module**.exports = mongoose;

**3.** Definir los esquemas. Se hace a través de la interfaz Shema.

Cada uno de los esquemas define la forma de los documentos dentro de una colección en MongoDB.

- Requerir el módulo de los modelos y la interfaz Shema.

**var** models = **require**('./models'),
Schema = models.Schema;

- Definir el esquema de la estructura de tu documento.

Ej.

**var** PostSchema = Schema({
content : 'string',
user : {
                 type : Schema.Types.ObjectId,
                 **ref** : 'user'
        }
});

Los tipos de esquemas permitidos son:

-   String
-   Number
-   Date
-   Buffer
-   Boolean
-   Mixed
-   ObjectId
-   Array

En [**http://mongoosejs.com/docs/schematypes.html**](http://mongoosejs.com/docs/schematypes.html) encontraras más sobre los schematypes permitidos.

**Nota:** Además de definir la estructura de los documentos y los tipos de datos que se están almacenando, los esquema se encargan de la definición de:

Validators (async and sync), Defaults, Indexes, Middlewares, Methods definition, Statics definition y Plugins

- Crear el modelo a partir del esquema; le pasas el nombre del modelo y el esquema.
Ej.

**var** Post = models.model('post', PostSchema);

Y como se está trabajando con módulos, para terminar hay que exportar el modelo.

Ej.

**module**.exports = Post;