UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Diplomatura en programación web full stack con React JS

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 4: NodeJS Intermedio

Unidad 3: Patrón MVC + Service

Presentación:

En este módulo vamos a aprender a aplicar a nuestro proyecto Express el patrón de diseño MVC que es uno de los más utilizados. De este modo, vamos a lograr estructurar mejor nuestro proyecto para que sea más fácil poder mantenerlo y detectar posibles errores.

Objetivos:

Que los participantes*:

Incorporen los conceptos de patrón de diseño, patrón de diseño MVC, Service y sean capaces de desarrollar una aplicación siguiendo este patrón.

Bloques temáticos:

1. Concepto de patrones de diseño
2. El patrón de diseño MVC
3. Concepto de Service
4. Implementación de MVC + Service en Express
5. Ejemplo
6. Trabajo Práctico

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

** El MEC es el modelo de E-learning colaborativo de nuestro Centro.*

Tomen nota;

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Concepto de patrones de diseño

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software, así que un patrón de diseño resulta ser una solución a un problema de diseño.

Objetivos de los patrones

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas de software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje a las nuevas generaciones de diseñadores condensando conocimiento ya existente.

No es obligatorio utilizar los patrones, solo es aconsejable.

Categorías de patrones

Según la escala o nivel de abstracción:

- **Patrones de arquitectura**: Aquellos que expresan un esquema organizativo estructural fundamental para sistemas de software.
- **Patrones de diseño**: Aquellos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software.
- **Dialectos**: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

Además, también es importante reseñar el concepto de "antipatrón de diseño", que con forma semejante a la de un patrón, intenta prevenir contra errores comunes de diseño en el software. La idea de los antipatrones es dar a conocer los problemas que acarrean

ciertos diseños muy frecuentes, para intentar evitar que diferentes sistemas acaben una y otra vez en el mismo callejón sin salida por haber cometido los mismos errores.

Material tomado de Wikipedia.
https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o

2. El patrón de diseño MVC

El patrón de diseño Modelo-Vista-Controlador es un patrón que separa los datos de la lógica de negocios de la aplicación. Propone la construcción de 3 componentes distintos que son el modelo, la vista y el controlador.

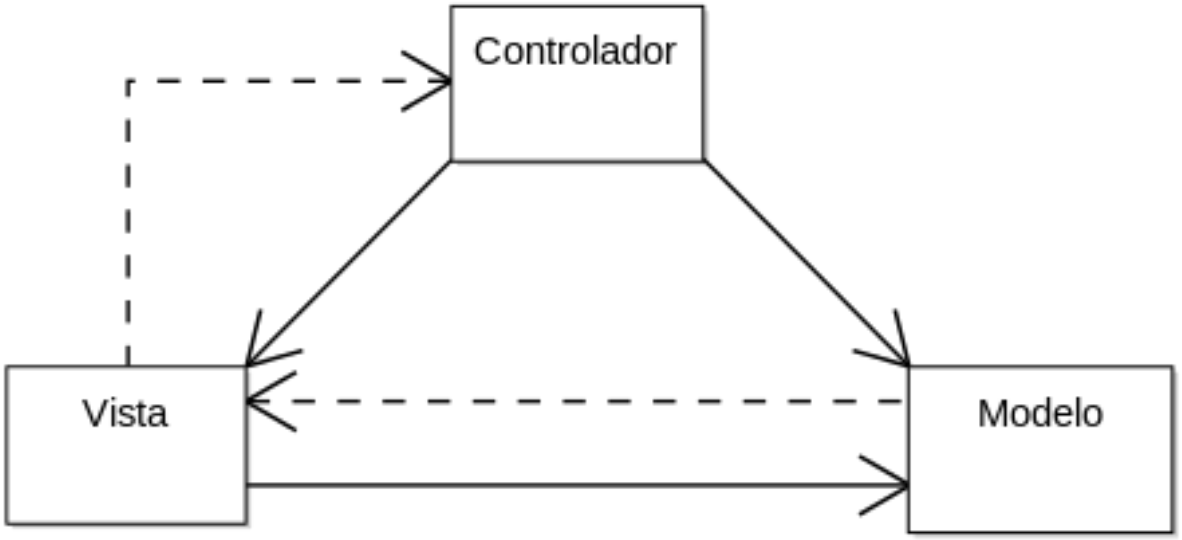
- **Modelo**: solo contiene datos, no contiene lógica.
- **Vista**: presenta al usuario los datos del modelo.
- **Controlador**: entre la vista y el modelo, escucha los sucesos y ejecuta la reacción apropiada

Cuando diseñamos una aplicación siguiendo el patrón de diseño MVC, vamos a crear 3 tipos diferentes de archivos. Se suele utilizar carpetas distintas para los modelos, las vistas y los controladores.

La capa de datos (modelo) contendrá un archivo por cada tabla de la base de datos. Allí se incluirán las consultas necesarias para la tabla en particular.

La capa de presentación (vista) “conoce” al modelo que va a mostrar, pero el modelo no “conoce” a la vista. La vista es invocada por el controlador con los datos que obtuvo del modelo.

La capa de lógica de negocio (controlador) tendrá la tarea de comunicarse con el exterior, aplicar la lógica de negocio, invocar a las vistas necesarias y devolverlas al cliente.



En cuanto a la vista, es menester aclarar que puede tratarse tanto de mensajes JSON por ejemplo, o bien templates creados con un motor como Handlebars, etc. La vista es la información a responder al cliente que la ha solicitado. La forma que tome dicha información depende de cada proyecto.

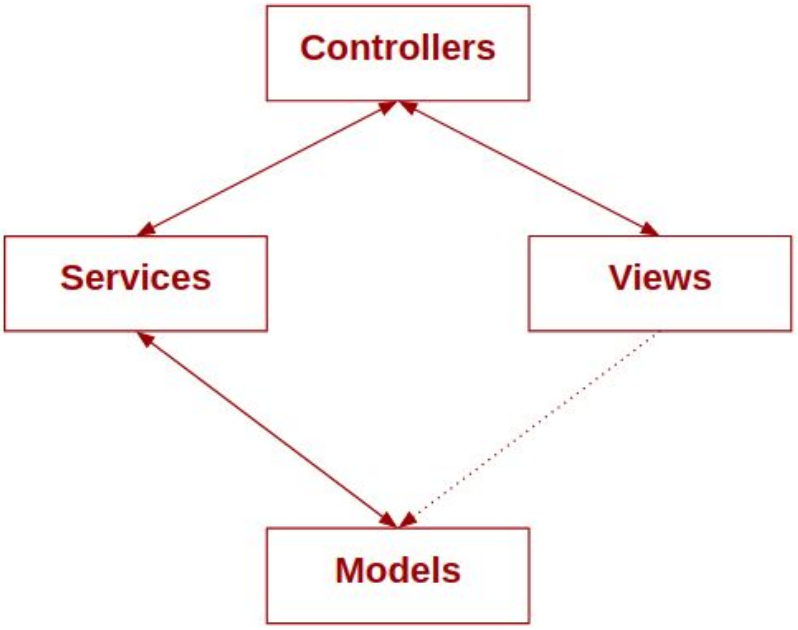
3. Concepto de Service

Haciendo un reduccionismo, los servicios son una forma de dividir lo que el patrón MVC incluye en el controller. Como vimos en secciones anteriores, el patrón MVC deja en manos del controller la lógica de negocio y la conexión de la aplicación con el exterior. En NodeJS, el controller sería el encargado tanto del ruteo como de la lógica del negocio. Sin embargo si bien el patrón MVC trae ventajas por la división que ofrece, aún queda demasiado en el controller. Los servicios vienen a mejorar la lógica del controller para dividirla y hacerla más mantenible.

Patrón MVC + Service significa agregar una capa extra para dividir la lógica del controller en dos partes:

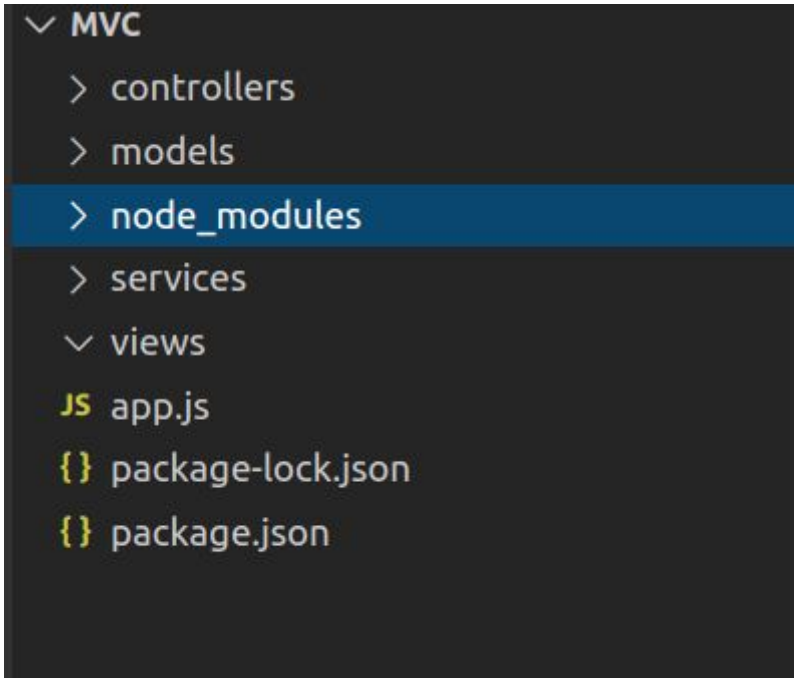
- Ruteo, conexión con el exterior, validación primaria de datos -> **Controller**
- Lógica de negocio, conexión con el model -> **Service**

Básicamente se desacopla la lógica de negocio de la lógica de conexión exterior.


Controllers
Services
Views
Models

4. Implementación de MVC + Service en Express

Para la implementación del patrón MVC + Services se sugiere la creación de un árbol de directorios (carpetas) para estructurar el proyecto.



Se sugiere la creación de una carpeta que contenga los controladores, otra para las vistas, una más para los models y finalmente una carpeta para los services. Quedando el app.js en la raíz del proyecto.

Models

Dentro de la carpeta models va la representación de cada una de las tablas de la base de datos y sus respectivas consultas.

Será un archivo por tabla. En caso de consultas que involucren más de una tabla, el

service se encargará de dicha lógica, invocando los modelos necesarios en el orden requerido para completar la consulta final.

Los modelos serán invocados por los services.

Views

Las vistas son las representaciones de la información que el sistema responde a sus clientes. Están íntimamente relacionadas con los modelos aunque no necesariamente haya una vista por cada modelo, en realidad, habrá tantas vistas como respuestas de información se requiera.

Las vistas pueden ser desde JSONs hasta motores de templates. Depende del proyecto la forma que tomarán las vistas. Por ejemplo, en un proyecto con cliente ReactJs, las vistas serán JSONs.

Las vistas pueden contener datos de más de un modelo, por ejemplo, si se requiere responder con la información de una factura, la vista contendrá tanto la información del encabezado de la factura (de un modelo) como los diferentes ítems de la factura (alojados en otro modelo).

Controllers

En proyectos pequeños todos los controllers podrían estar contenidos dentro del app.js En proyectos un poco más grandes es conveniente crear una carpeta y hacer un controller por cada ruta.

Los controllers van a encargarse de hacer una primera validación de la información recibida. Se orienta a verificar que se hayan recibido todos los parámetros necesarios y que los mismos contengan el tipo de dato requerido. El resto de las validaciones referidas a la lógica de negocio se realizan en los services.

Serán los controllers los encargados también de enviarle a las vistas los datos obtenidos

de los services y responder con la respuesta de las vistas al cliente que realizó la petición.

Al crear los controladores, se suele utilizar como parte del nombre la palabra “Controller”, por ejemplo personaController.js

Services

Por lo general, habrá uno por modelo aunque es posible que haya más. Los services contienen la lógica de negocio de la aplicación que involucra:

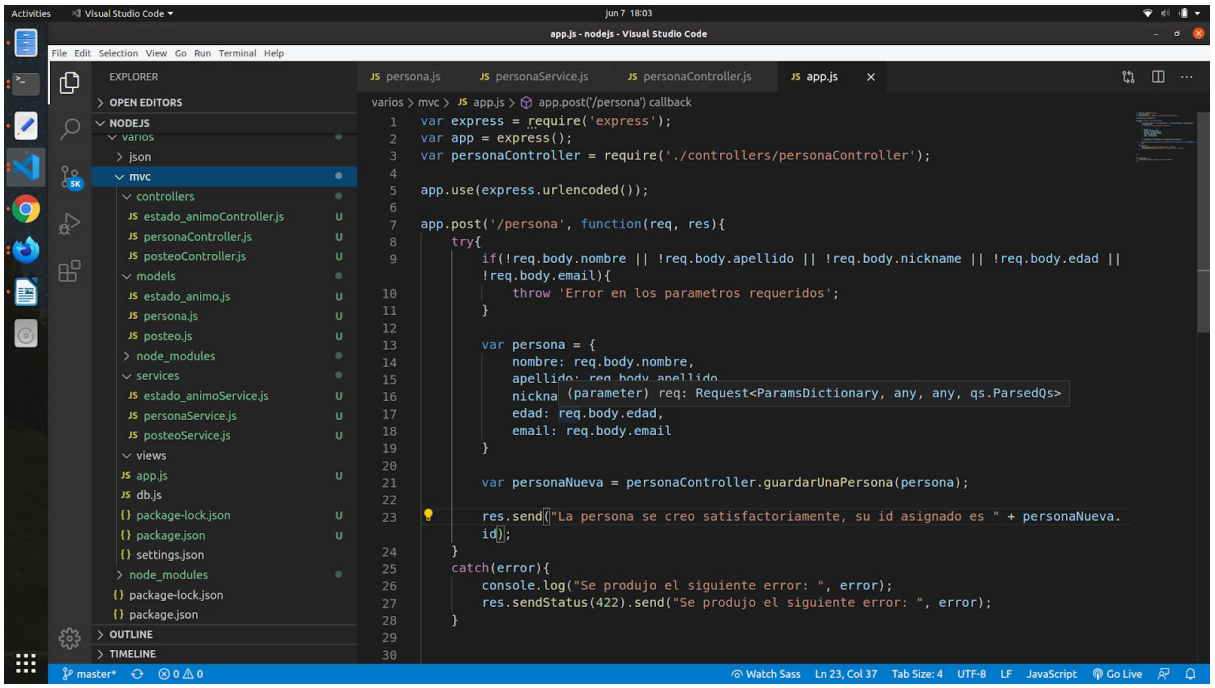
+ Validación de los datos de entrada conforme lo requiera el proyecto, por ejemplo, verificar que los CUIT (Clave Unica de Identificación Tributaria Argentina) sean válidos (existe un algoritmo para ello). Y vale la aclaración que mientras el controller verificará que este campo llegue, tenga datos y que esos datos corresponden al formato de CUIT (nn-nnnnnnnn-n), será el service el encargado de comprobar que dicho valor cumpla las especificaciones de la fórmula de CUIT.
+ Procesamiento de los datos recibidos.
+ Consultas a los modelos necesarios.
+ Procesamiento de las respuesta recibidas de las consultas
+ Retorno de los datos al controlador que invoco al service.

Al crear un service, se suele agregar al nombre la palabra “Service”, por ejemplo, personaService.js

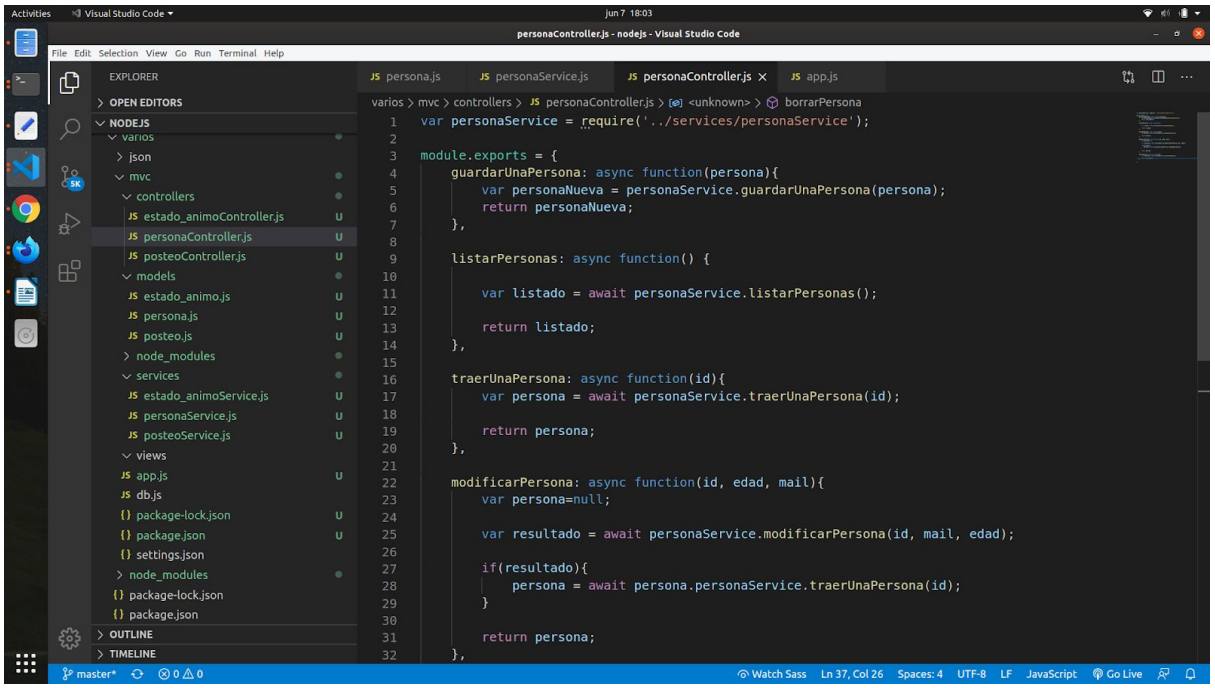
5. Ejemplo

El ejemplo contiene la implementación vertical de una funcionalidad de una aplicación de posteos al estilo twitter donde una persona que haya ingresado previamente sus datos al sistema, puede realizar posteos indicando su nombre, su estado de ánimo y lo que quiera contar en 240 caracteres.

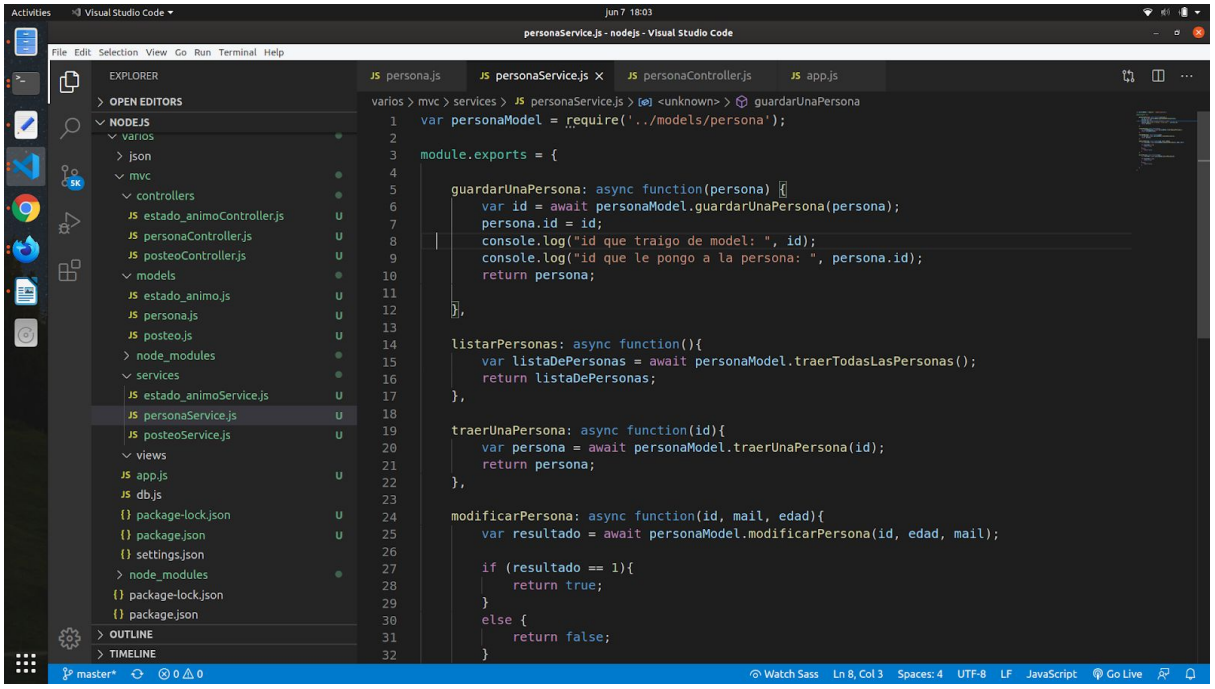
Estructura de un servidor NodeJS implementando MVC + Service



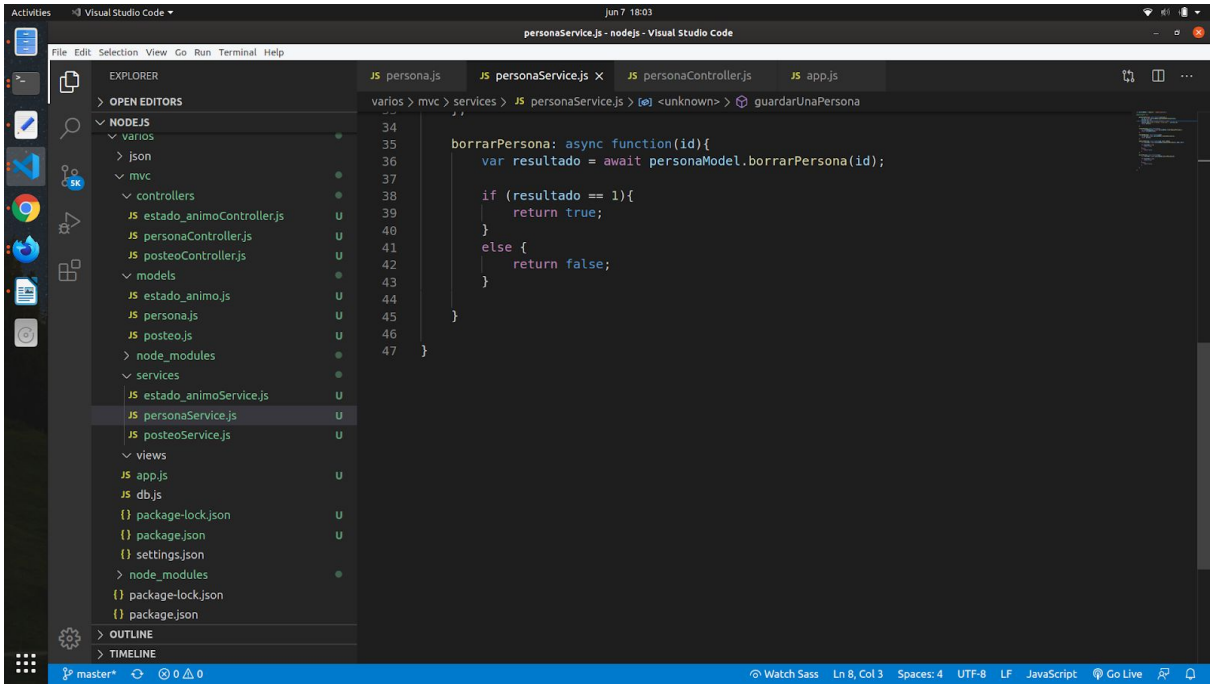
En app.js queda la lógica de ruteo. Aunque no está representada en la imagen, la función anónima también debe ser asincrónica y la llamada al controller debe contener el await correspondiente.

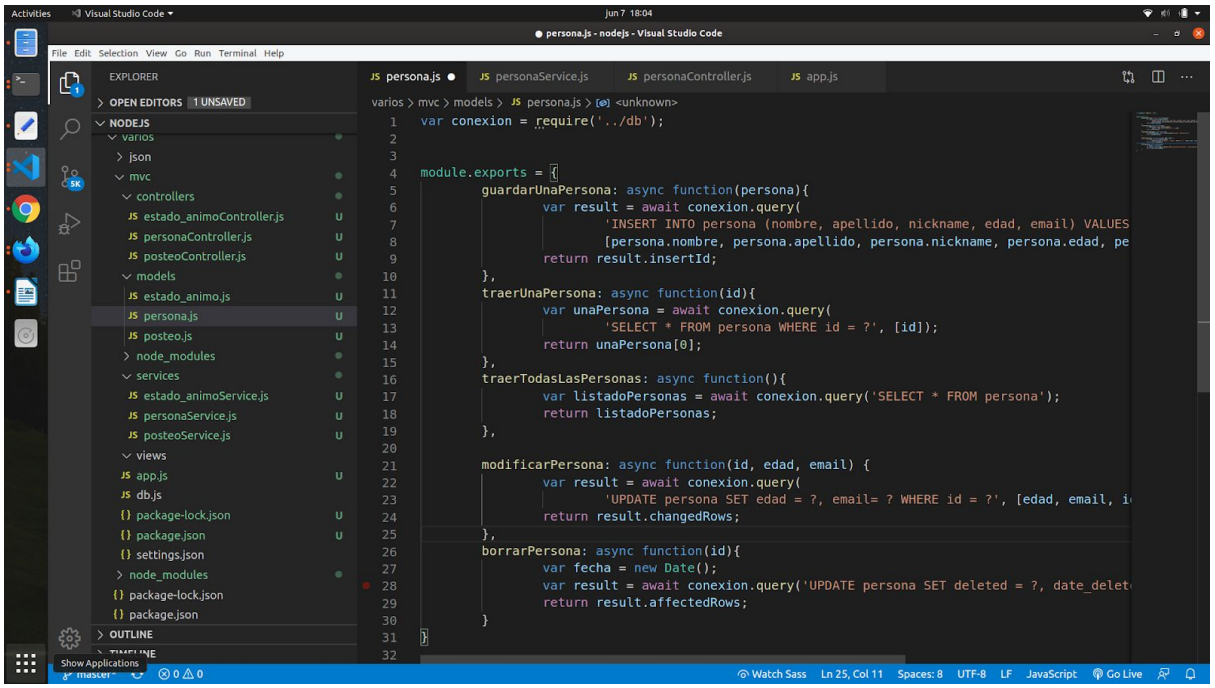


El controller se encarga de verificar la información que será enviada a los services. Verifica que los datos obtenidos cumplan con la lógica de negocios.

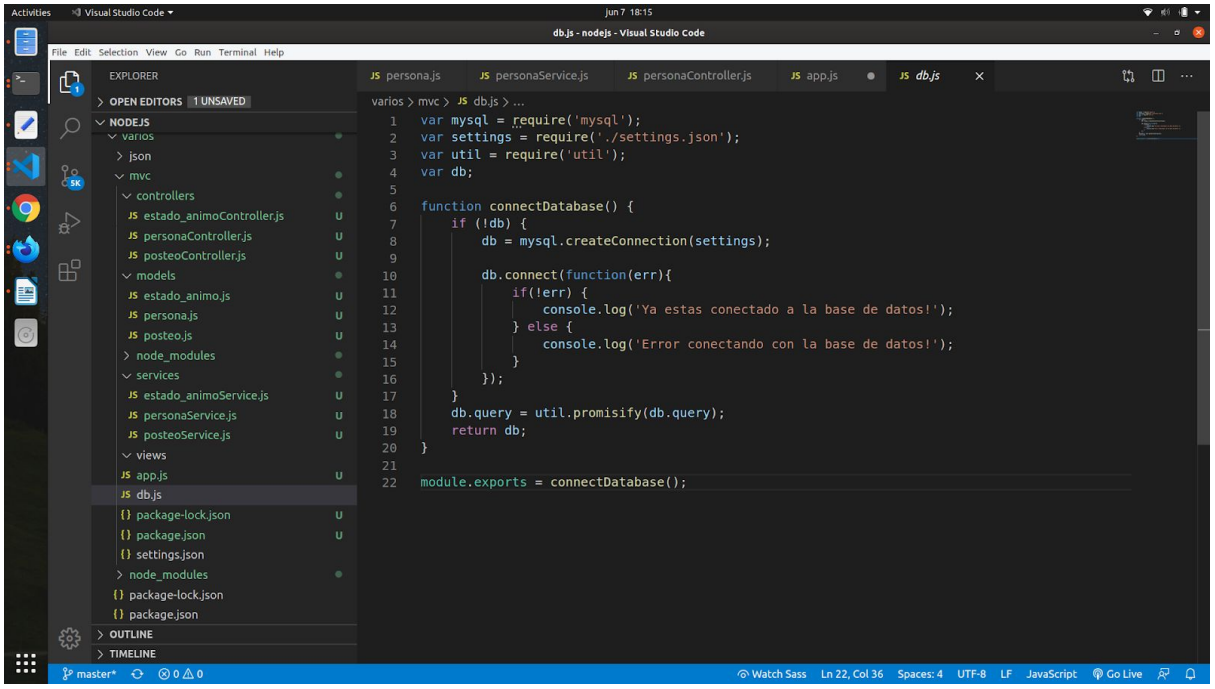


En el service se implementa la lógica de negocios más profunda, se trabaja con los models y se responde a los controllers. Si fuera requerido realizar cálculos o transformaciones de datos, aquí sería el lugar donde sucedería.





El modelo realiza las consultas a la base de datos y devuelve la información.



La conexión a la base de datos se realiza en archivo aparte que luego es incluída en los diferentes modelos.

6. Trabajo Práctico

Tomando de base el ejemplo, realizar la implementación del resto de los recursos para completar el servidor del ejemplo.

Bibliografía utilizada y sugerida

Express (n. d.) Recuperado de: https://expressjs.com/es/

MDN - JavaScript (n.d.) Recuperado de:

https://developer.mozilla.org/es/docs/Web/JavaScript

NodeJs (n. d.) Recuperado de: https://nodejs.org/es/

NodeJS Documentacion (n. d.) Recuperado de: https://nodejs.org/es/docs/

NPM (n. d.) Recuperado de: https://www.npmjs.com/package/page

Wikipedia - NodeJS (n. d.) Recuperado de: https://es.wikipedia.org/wiki/Node.js

Lo que vimos:

Patrón de diseño MVC + Service.

Lo que viene:

Testeo y deploy.