**Notas del curso**

**02. Workflow and debugging**

**1. Uso de npm**

npm init

npm install <package\_name>

npm install --save-dev nodemon

**2. Uso de nodemon**

npm start

npm run <script\_name> //Para custom scripts

**3. Debugging Node.js app con VSC**

- Config. extra:

#En el archivo '.vscode/launch.json' añadir:

#Permitirá reiniciar automáticamente el debugger

"restart": true,

"runtimeExecutable": "nodemon",

"console": "integratedTerminal"

**03. Express,js**

-**Middleware**

Se ejecuta de forma secuencial según el orden que exista en el código.Cualquier ruta irá a la '/' por lo que otra ruta debe ir antes del root. Para que la request continúe al sifg. middleware debe de usarse Next().

**- BodyParser**

**- Router de Express**

**- Filtering paths**

**- Servir contenido estáticamente**

app.use(express.static(path.join(\_\_dirname, "public")));

**04. Dynamic Content & Templating engines**

**pug**

**handlebars - hbs**

**ejs**

**Express.js app.set()**

Mantener variables de forma global en la aplicación.

En esta sección se vieron los anteriores *templating engines* y al final se limipió todo y sólo se dejó lo de *ejs*. Por lo que la carpeta *04 MVC* está limpia y sólo se usa *ejs*.

**05. The Model View Controller**

En esta sección el proyecto fue convertido al paradigma de Modelo Vista Controlador (MVC), se creó un modelo y controladores para las rutas existentes.

**06. Optional: Enhancing the app**

En esta sección se agregaron más características a la tienda online, como más atributos al modelo, más rutas, controladores y vistas para incrementar las funcionalidades de la tienda online.

**07. Dynamic routes & Advanced models**

**Passing route params**

**Using query params**

**?param=value&isActive=true**

Example:

/products/:productId?edit=true

/\* Access to the data \*/

//Params

req.params.productId

//Parameters

req.query.edit

**Enhance our models**

Se trabajó con métodos estáticos para no tener que crear varias instancias de un producto.

Se puede interactuar entre *models*.

Carpeta: 05

**08. SQL introduction**

Se aprendió a cómo conectar MySQL a la aplicación Node.js, se ejecutó el sig. comando:

npm install –-save mysql2

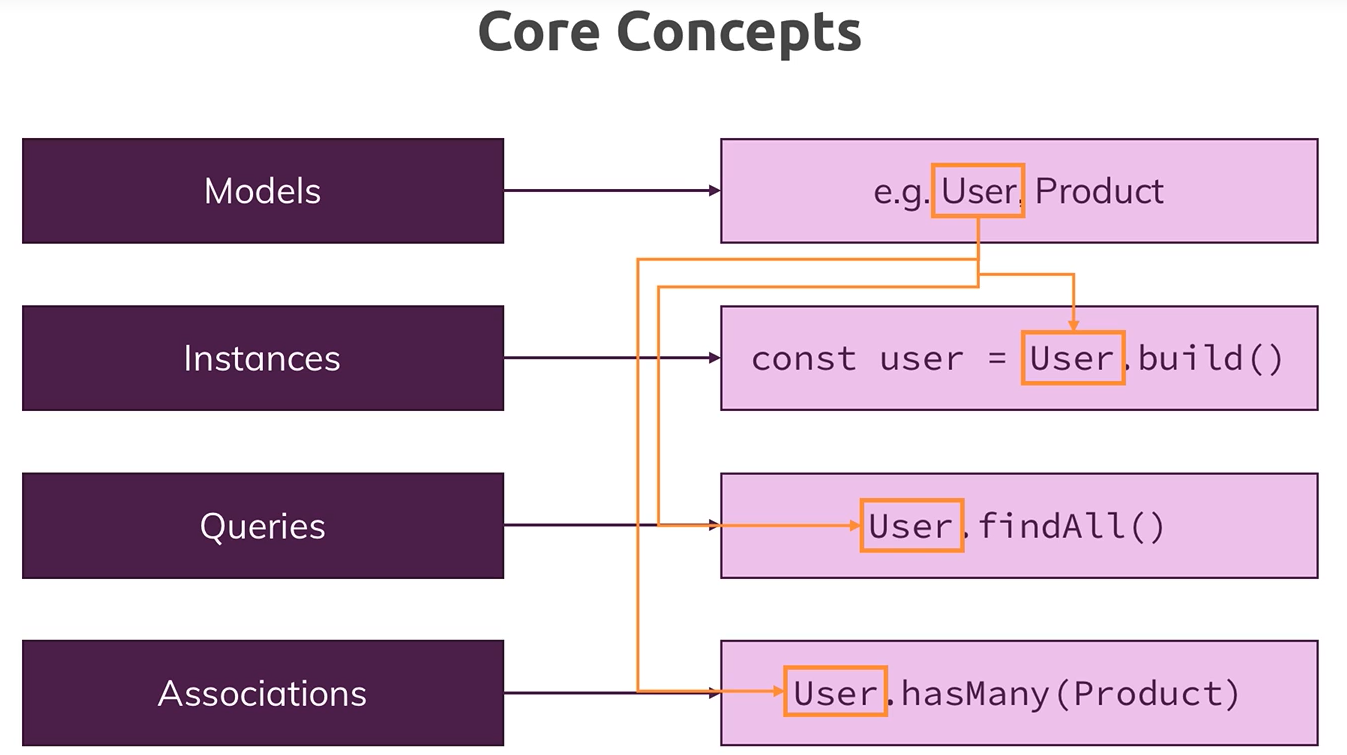
Se creó una base de datos llamada *nodejs*-*course* y una tabla llamada *products*, se hizo una conexión de tipo *pool* para conexiones persistentes sin necesidad de estar abriendo y cerrando la conexión. Se ejecutaron sentencias básicas para insertar y recuperar datos.

Se modificaron los modelos para trabajar con la BD en vez de archivos de texto.

Carpeta: 05, 06

**09. Sequelize**

Es un ORM – Object-Relational Mapping Library



**Instalación**

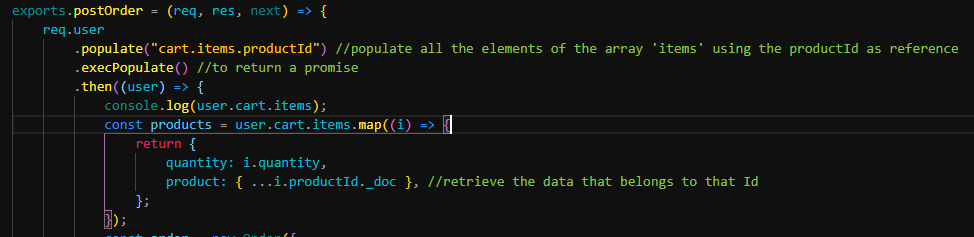
npm install –-save mysql2 //required

npm install --save sequelize

**10. NoSQL using MongoDB**

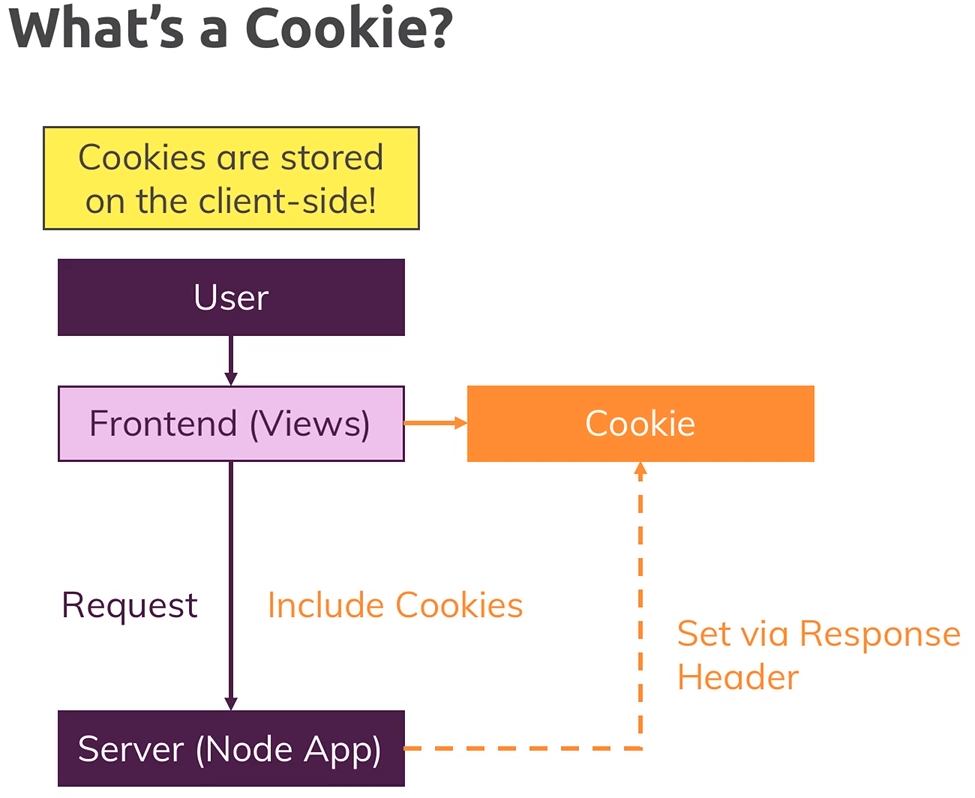
En esta sección se utilizó MongoDB como base de datos, fue más sencillo que utilizar sequelize, sin embargo, prefiero más mongoose porque es más sencillo y tiene métodos mejores para trabajar con la información.

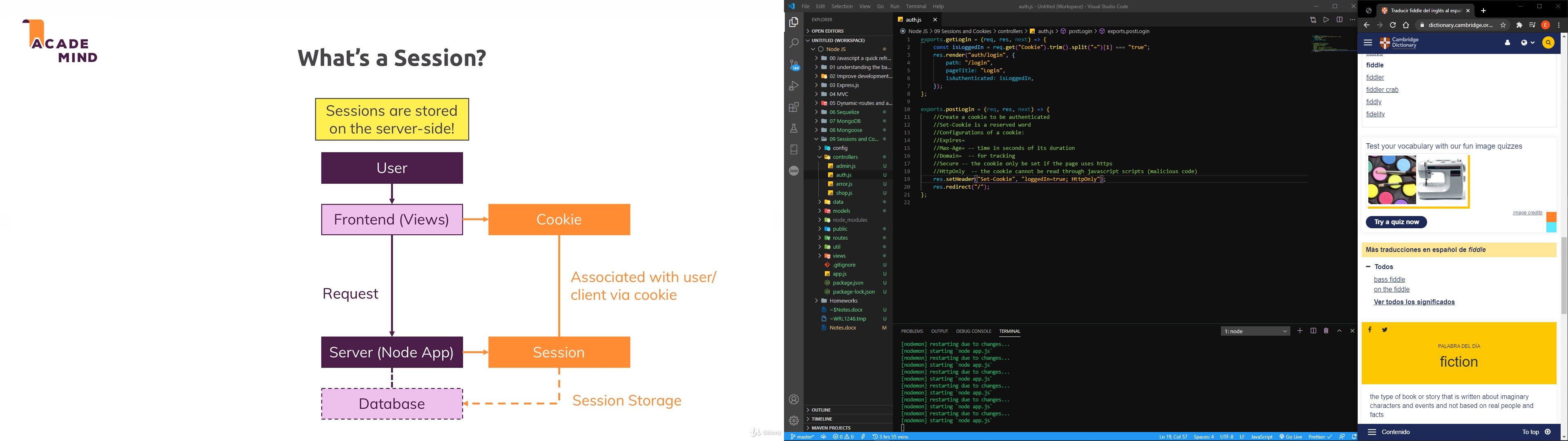
Aún siento que la aplicación que se está creando no es bastante profesional, y aún se utilizan dummy values.

**11. Mongoose**

En esta sección se aprendió *mongoose* que realmente es bastante útil. Sería muy bueno leer la documentación, tiene bastantes cosas de utilidad.

**12. Sessions & Cookies**



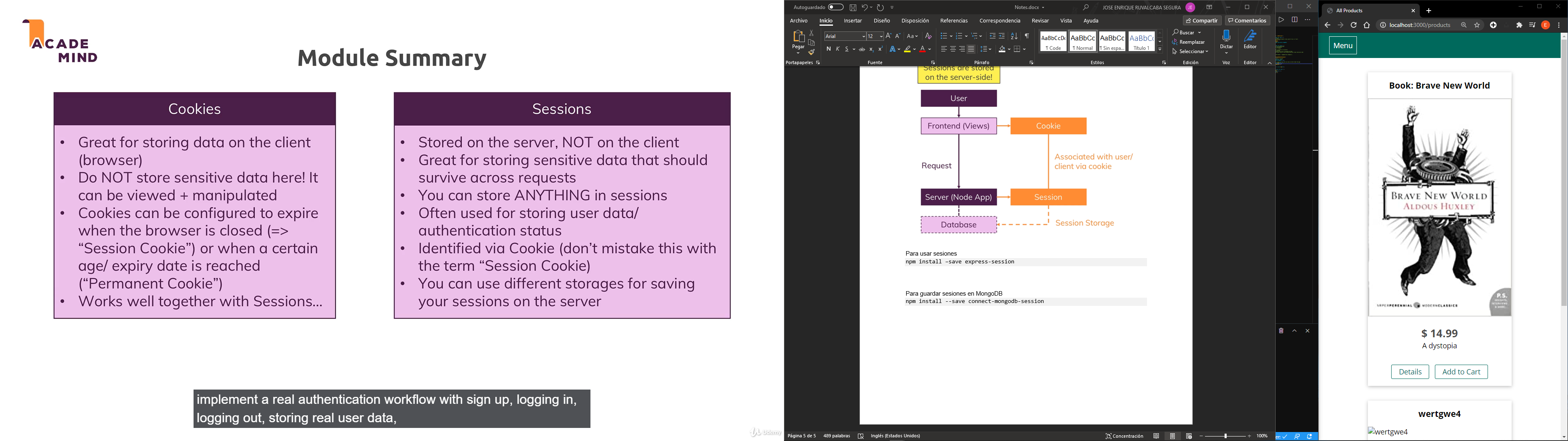


Para usar sesiones

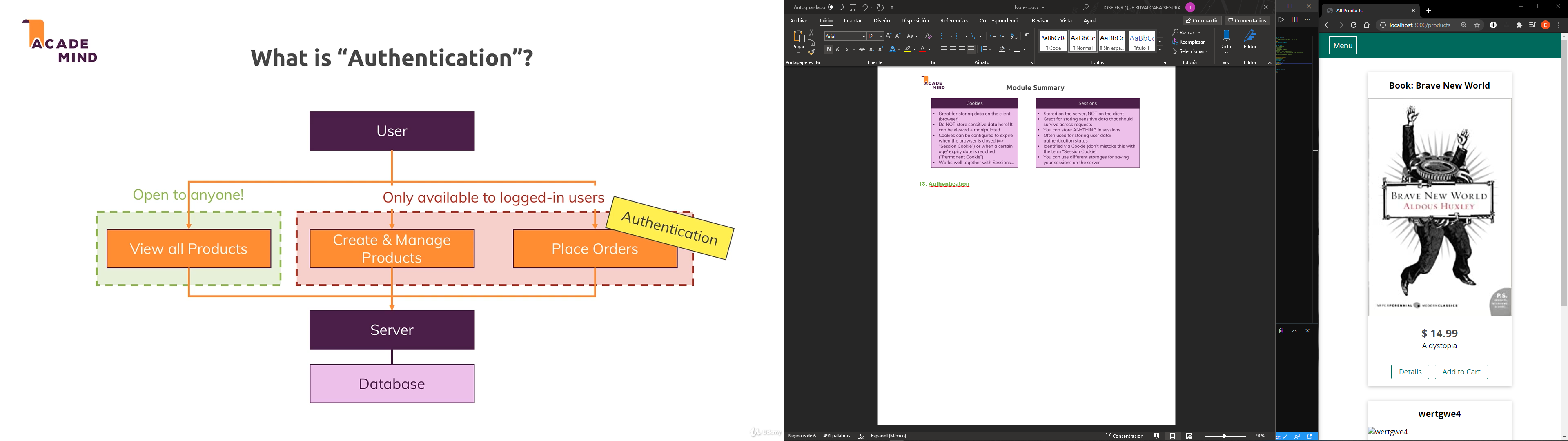
npm install –save express-session

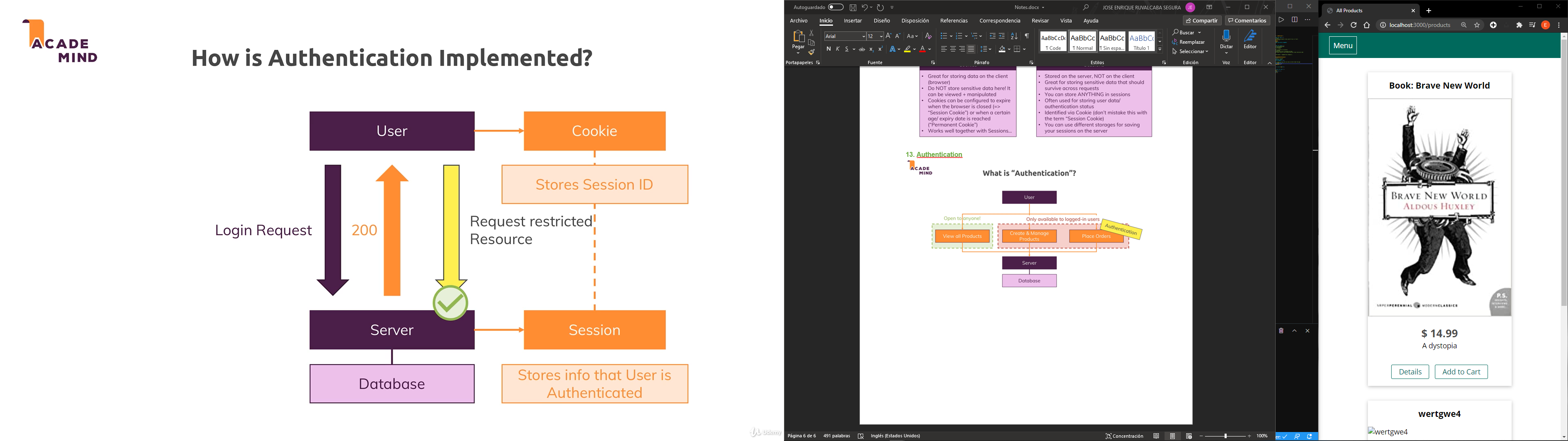
Para guardar sesiones en MongoDB

npm install --save connect-mongodb-session



**13. Authentication**

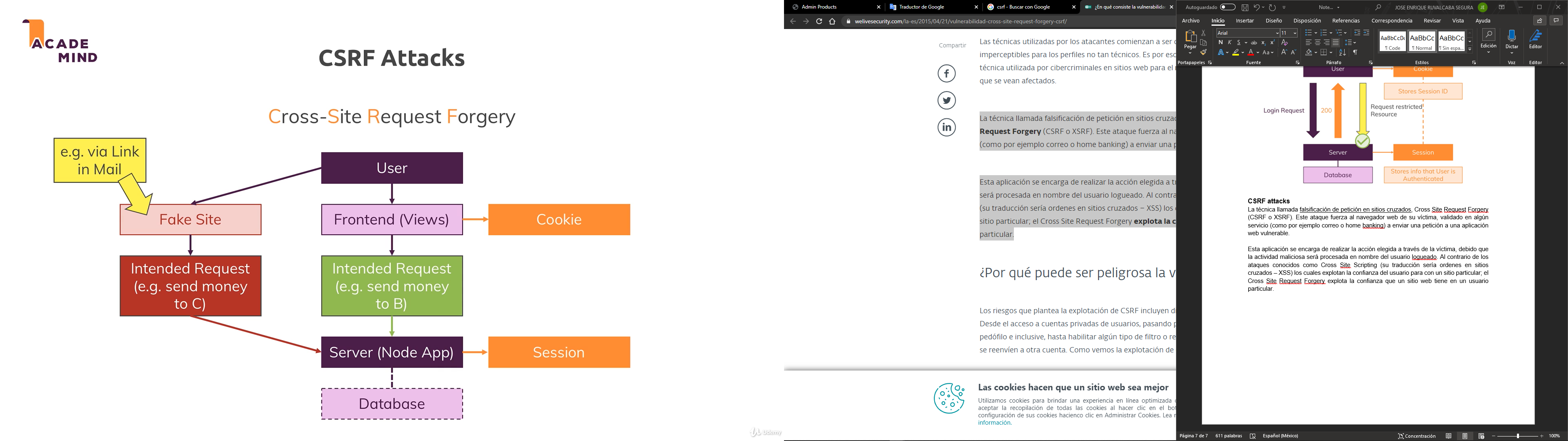




**CSRF attacks**

La técnica llamada falsificación de petición en sitios cruzados, Cross Site Request Forgery (CSRF o XSRF). Este ataque fuerza al navegador web de su víctima, validado en algún servicio (como por ejemplo correo o home banking) a enviar una petición a una aplicación web vulnerable.

Esta aplicación se encarga de realizar la acción elegida a través de la víctima, debido que la actividad maliciosa será procesada en nombre del usuario logueado. Al contrario de los ataques conocidos como Cross Site Scripting (su traducción sería ordenes en sitios cruzados – XSS) los cuales explotan la confianza del usuario para con un sitio particular; el Cross Site Request Forgery explota la confianza que un sitio web tiene en un usuario particular.



**Forma de proteger la aplicación de ataques CSRF**

* CSRF Token

**CSRF Token**

A CSRF token is a unique, secret, unpredictable value that is generated by the server-side application and transmitted to the client in such a way that it is included in a subsequent HTTP request made by the client. When the later request is made, the server-side application validates that the request includes the expected token and rejects the request if the token is missing or invalid.

CSRF tokens can prevent CSRF attacks by making it impossible for an attacker to construct a fully valid HTTP request suitable for feeding to a victim user. Since the attacker cannot determine or predict the value of a user's CSRF token, they cannot construct a request with all the parameters that are necessary for the application to honor the request.

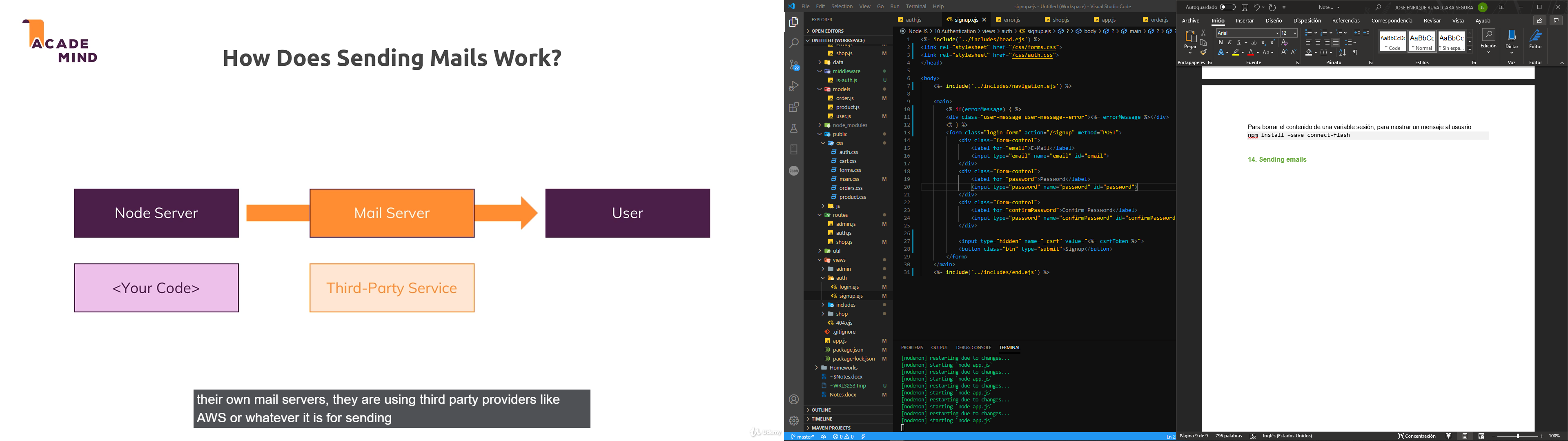
npm install -–save csurf

Al agregar la protección en el server, csrf buscará el token csrf válido para cada petición POST que sea realizada.

Para borrar el contenido de una variable sesión, para mostrar un mensaje al usuario

npm install –save connect-flash

**14. Sending emails**



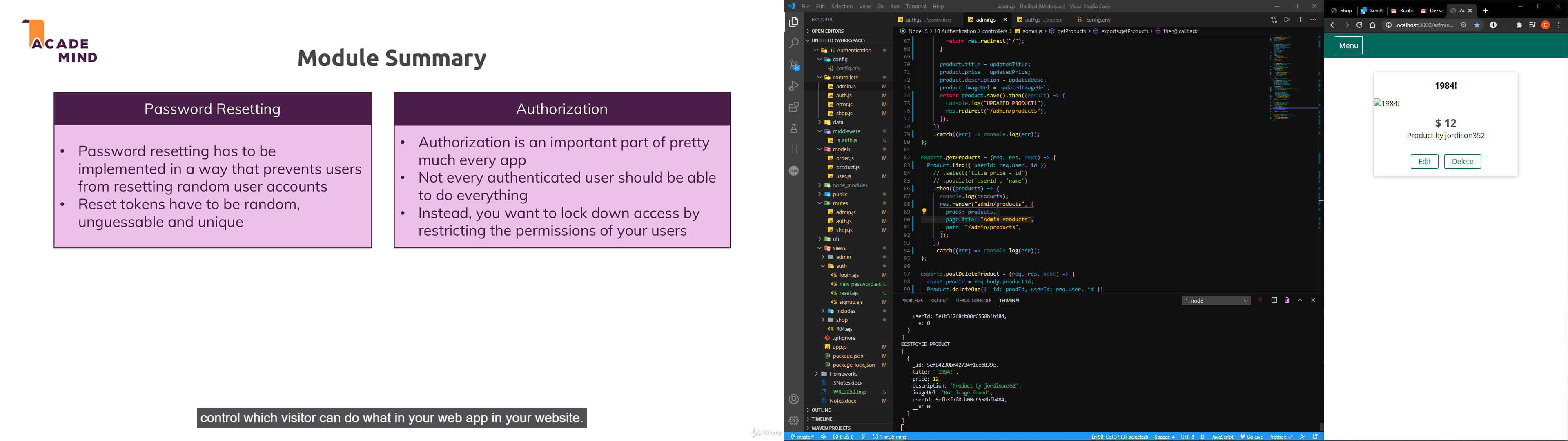
**Email servers**

* SendGrid
* MailChimp
* AWS
* SCS
* Nodemailing

npm install --save nodemailer

npm install --save nodemailer-sendgrid-transport

**15. Advanced authentication**



**16. Understanding validation**

npm install --save express-validator

*Check* permite validar valores del *body, params o body*.

También pueden usarse de forma específica.

const { check, body, param, query, header } = require("express-validator");

**16. Error handling**

En esta sección se vieron las formas de manejar errores, en la cual se usó principalmente el que tiene incluido *express.* Hay distintas formas de lanzar errores, varía según si el error está dentro de una función *async* o una que es *sync*.

*Controller*

.catch((err) => {

      const error = new Error(err);

      error.httpStatusCode = 500;

      return next(error);

    });

*Middleware*

app.use((error, req, res, next) => {

  //res.status(error.httpStatusCode).redirect("/500");

  res

    .status(error.httpStatusCode || 500)

    .render(error.httpStatusCode || "500", {

      pageTitle: "Error ocurred!",

      path: "/500",

      isAuthenticated: req.session.isLoggedIn,

    });

});

**17. File uploads & downloads**

npm install --save multer

Multer permite parsear requests para archivos.

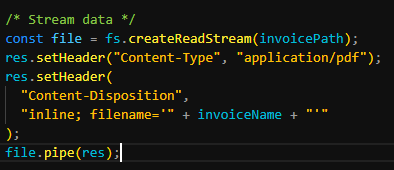
Enctype multipart le dice al server que el submit no sólo será texto plano sino distintos tipos de datos.

<form class="product-form" method="POST" enctype="multipart/form-data">

**Preload data vs Stream data**

Preload data: Para archivos pequeños.

Stream data: Archivos grandes. (Recomendado)



**Crear archivos pdf**

npm install --save pdfkit

**18. Pagination**

*skip* y *limit* permiten devolver documentos de la base por partes

 Product.find()

    .countDocuments()

    .then((numProducts) => {

      totalItems = numProducts;

      return Product.find()

        .skip((page - 1) \* ITEMS\_PER\_PAGE) //skip the first two items

        .limit(ITEMS\_PER\_PAGE); //only show two items per page

    })

    .then((products) => {

      res.render("shop/product-list", {

        prods: products,

        pageTitle: "Products",

        path: "/products",

        currentPage: page,

        hasNextPage: ITEMS\_PER\_PAGE \* page < totalItems, //check if there are more products to show

        hasPreviousPage: page > 1, //check if could be a previous page

        nextPage: page + 1,

        previousPage: page - 1,

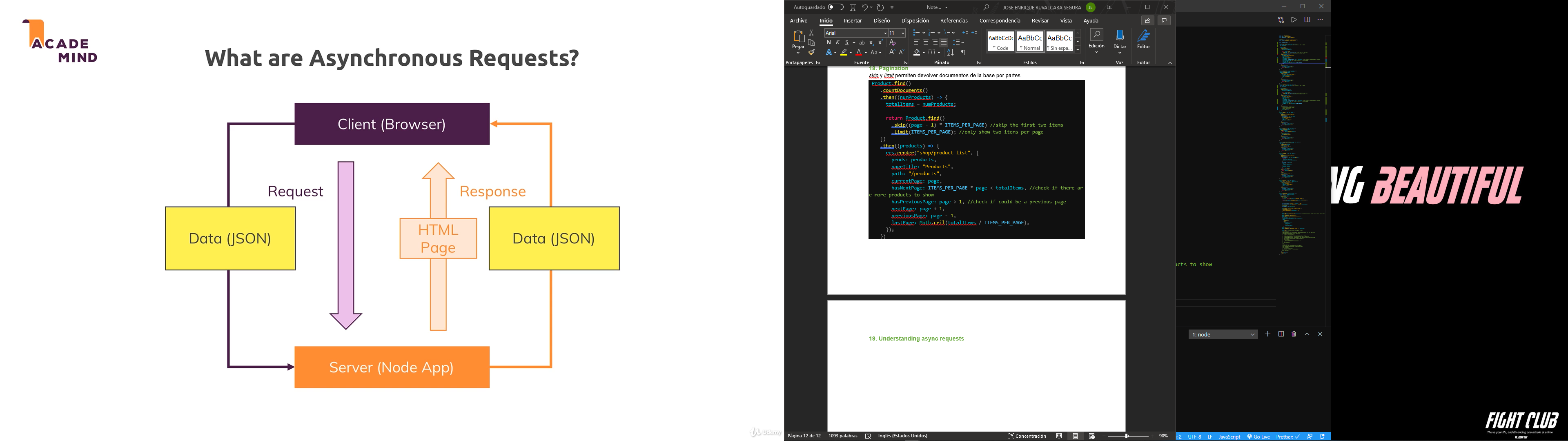
        lastPage: Math.ceil(totalItems / ITEMS\_PER\_PAGE),

      });

    })

**19. Understanding async requests**

Este tipo de *request* hace que únicamente se envíe datos en *json* en lugar de recargar o dirigir a una página, por lo que mejora la experiencia de usuario.



Para ello se eliminó el *form* en la *view* porque ya no es necesario enviar un *POST request*. El *type* del botón cambió a *button* y se mandó llamar una función que realiza un *DELETE request* al server.

El código *js* del lado del servidor es:

const deleteProduct = (btn) => {

  //access to the surrounding nodes

  const prodId = btn.parentNode.querySelector("[name=productId]").value;

  const csrf = btn.parentNode.querySelector("[name=\_csrf]").value;

  //Get the closest parent article of the button

  const productElement = btn.closest("article");

Se accede a los nodos.

Luego se usa fetch para enviar la petición. En el header se pasa el csrf token.

//fetch is not only for fetching data but sending data too

  fetch("/admin/product/" + prodId, {

    method: "DELETE",

    headers: {

      "csrf-token": csrf,

    },

  })

    .then((result) => {

      console.log(result);

      return result.json();

    })

    .then((data) => {

      console.log(data);

      //Supported for all the browsers

      productElement.parentNode.removeChild(productElement);

      //productElement.remove(); not supported in Internet Explorer

    })

Luego al final se elimina el producto de la view.

El código en la view es:

<input type="hidden" value="<%= product.\_id %>" name="productId">

<input type="hidden" name="\_csrf" value="<%= csrfToken %>">

<!-- 'this' is to pass the button itself-->

<button class="btn" type="button" onclick="deleteProduct(this)">Delete</button>

Notar que el botón se pasa así mismo con *this*.

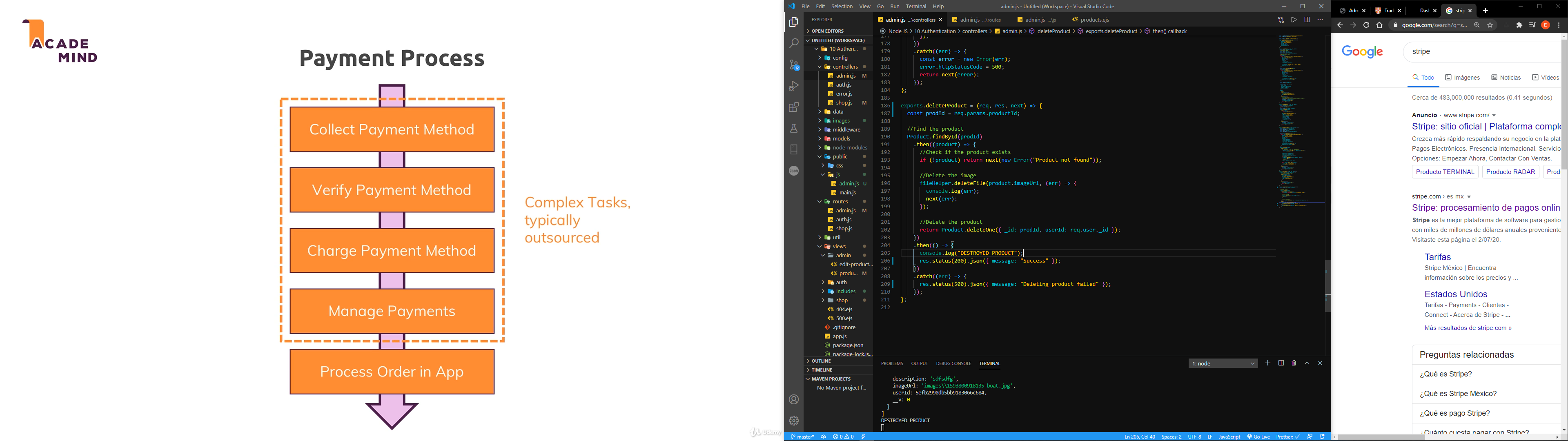
En la ruta el método pasó de ser POST a DELETE.

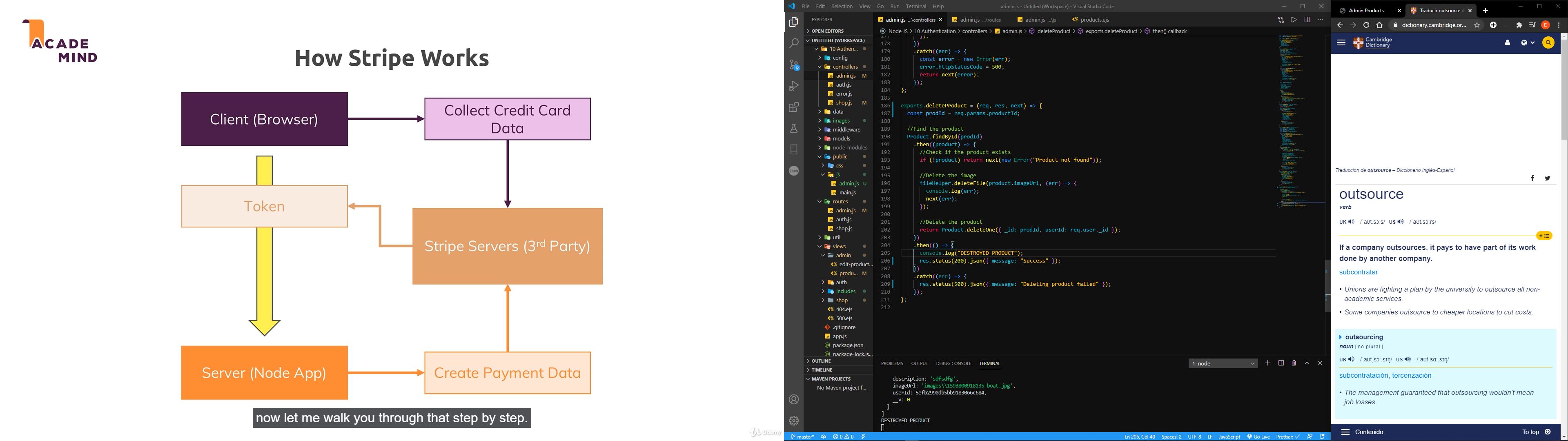
router.delete("/product/:productId", isAuth, adminController.deleteProduct);

Y al final el servidor envía una respuesta después de eliminar en la BD.

      res.status(200).json({ message: "Success" });

**20. Adding payments**





npm install --save stripe

Se aprendió cómo realizar pago con Stripe, sin embargo, la forma implementada en la app tiene un issue, el cual al dirigirse a la página de checkout sucess se genera una nueva orden según lo que contenga el carro, y aunque en Stripe no se ve reflejado, en la app sí. Existen varias formas de arreglarse eso, checar documentación.

<https://stripe.com/docs/payments/handling-payment-events?lang=node>

**21. REST APIs the basics**