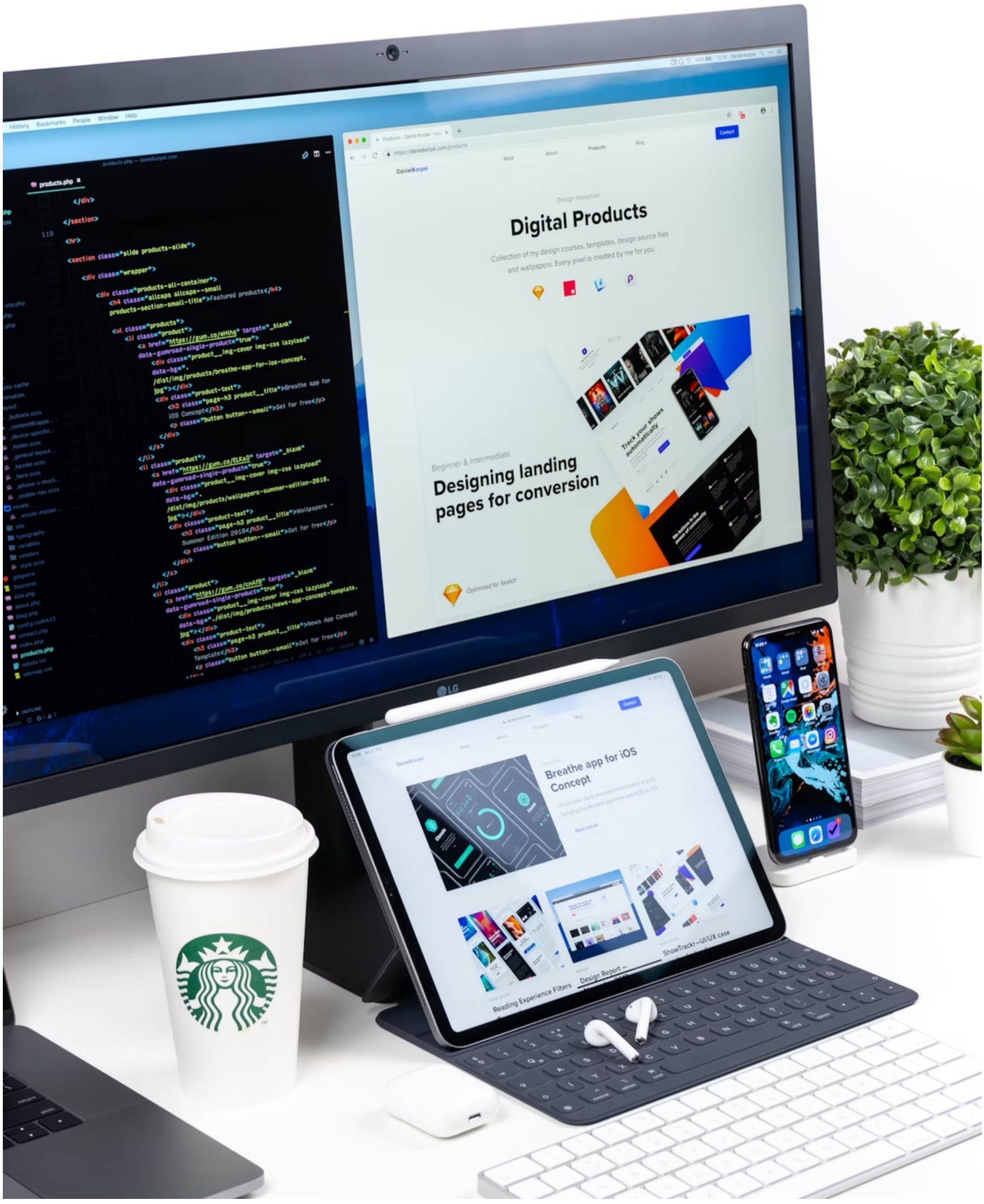
Lección 2:
Estructura de un servidor de aplicativo web



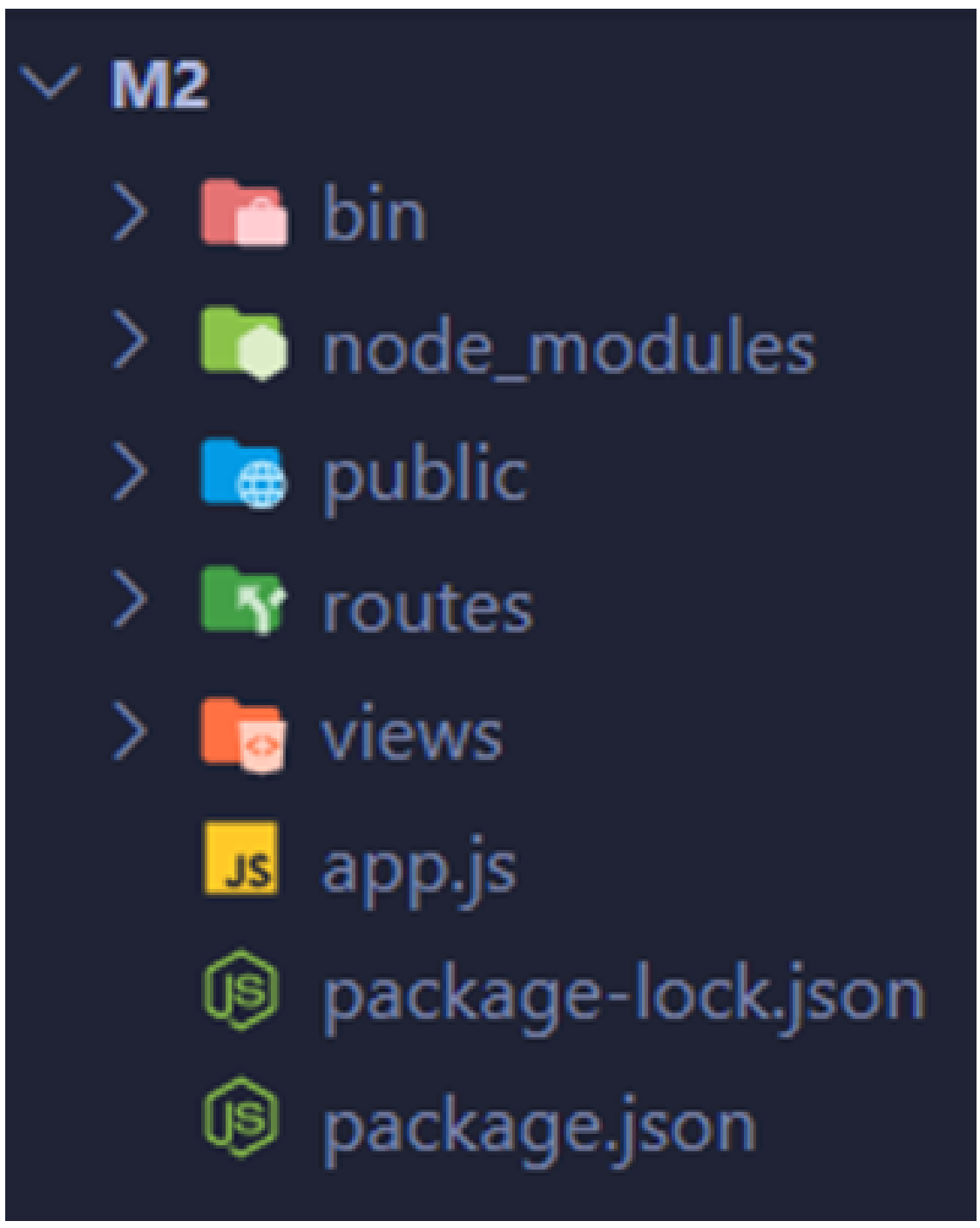
Planteamiento de la sesión

Materiales:

- Direccionamiento de Express
- File and folder structure for Node-Express applications | Medium
- GET vs. POST: en qué se diferencian los dos métodos de petición HTTP - IONOS
- Métodos de petición HTTP

El servidor web

Se ha creado un proyecto utilizando Express Generator y el resultado es una carpeta como la siguiente, muy diferente a los archivos iniciales del primer proyecto de Express creado.



Esto es debido a que el generador distribuye o crea una estructura base de carpetas que puede ser utilizada para aplicar el patrón MVC, sin embargo, como se mencionó en un inicio, MVC no sólo cuenta con las carpetas para los modelos, vistas y controladores, sino que hay una serie de elementos adicionales que hacen parte del servidor web.

Antes de conocer las carpetas adicionales, se explicarán los diferentes archivos y carpetas con las que cuenta el proyecto.

-En la carpeta raíz encontramos el entry point **app.js**, y el archivo **package.json** con las dependencias del proyecto.
-Dentro de la carpeta **bin** encontramos el archivo **www** sin extensión. El mismo trae definida una lógica interna y se encargará de hacer que la aplicación corra, es decir que en este archivo es donde se crea el servidor y se deja escuchando las peticiones.
-Dentro de la carpeta **public** podremos guardar todos los recursos **estáticos** de nuestra aplicación: **CSS**, **imágenes, JavaScript de frontend.**
-Dentro de la carpeta routes estaremos administrando el sistema de ruteo de la aplicación. Encontramos los archivos **index.js y users.js**.
-Dentro de la carpeta **views** encontramos dos vistas iniciales que trae el generador: **index.ejs y error.ejs**.
-Pendiente la creación de una carpeta llamada controllers. Esta carpeta contiene los controladores de la aplicación, son bloques de código que permiten conectar las vistas con los modelos y contienen parte de la lógica que tramita la petición del usuario.
-Pendiente la creación de una carpeta llamada **models**. La carpeta para los modelos guarda una representación de la estructura de cada tabla en la base de datos, sus campos, atributos y relaciones.
-y pendiente, por ahora, la creación de una carpeta llamada middlewares. Los **middlewares** son funciones que se ejecutan en medio de la ruta y el controlador que permite hacer logs de información o validaciones, según lo que sea necesario.
-Otra posible carpeta pendiente que se podría agregar es una llamada **helpers**. En esta carpeta se guardan funciones que pueden ser utilizadas a lo largo de diferentes partes del proyecto, así se pueden reutilizar sin necesidad de escribirlas una y otra vez.

Aunque existen diferentes formas de de crear los ficheros y agrupar los archivos, hay algunos principios que nos pueden guiar a la hora de tomar decisiones.

Principios de LIFT: Localize, Identify, Flat y Try-DRY Estos principios ayudan a organizar el código de forma coherente, clara y concisa.

-Localize: Agrupar de forma coherente
-Identify: Nombrar para indicar el contenido
-Flat: Crear subcarpetas solo en caso de necesidad
-Try-DRY: Cierta redundancia puede ser beneficiosa. DRY hace referencia a Don’t Repeat Yourself.

Las carpetas de primer nivel deben reflejar que se trata de un servidor de Express.

-/api
-/middleware
-/helpers
-/controllers
-/views
-/models

Qa<

```
├── node_modules ← Carpeta de módulos locales de proyecto
├── public ← Carpeta de archivos públicos
├── src ← Carpeta de archivos funcionales de la aplicación
│   ├── controllers
│   │   └── (acá van los controladores)
│   ├── views
│   │   └── (acá van las vistas)
│   ├── routes
│   │   └── (acá van las rutas)
│   └── app.js ← Archivo de entrada del proyecto
├── package.json
└── package-lock.json
```

Este modelo requiere modificar algunas líneas ya que el entry point o archivo principal del proyecto se encuentra en un nivel diferente a la raíz del mismo, más allá de esto, la estructura que se lleve dentro de /src o a nivel de la raíz del proyecto, debe cumplir, o al menos buscar hacerlo, con los principios mencionados para una distribución con sentido de los ficheros y directorios.

Recorrido de una petición

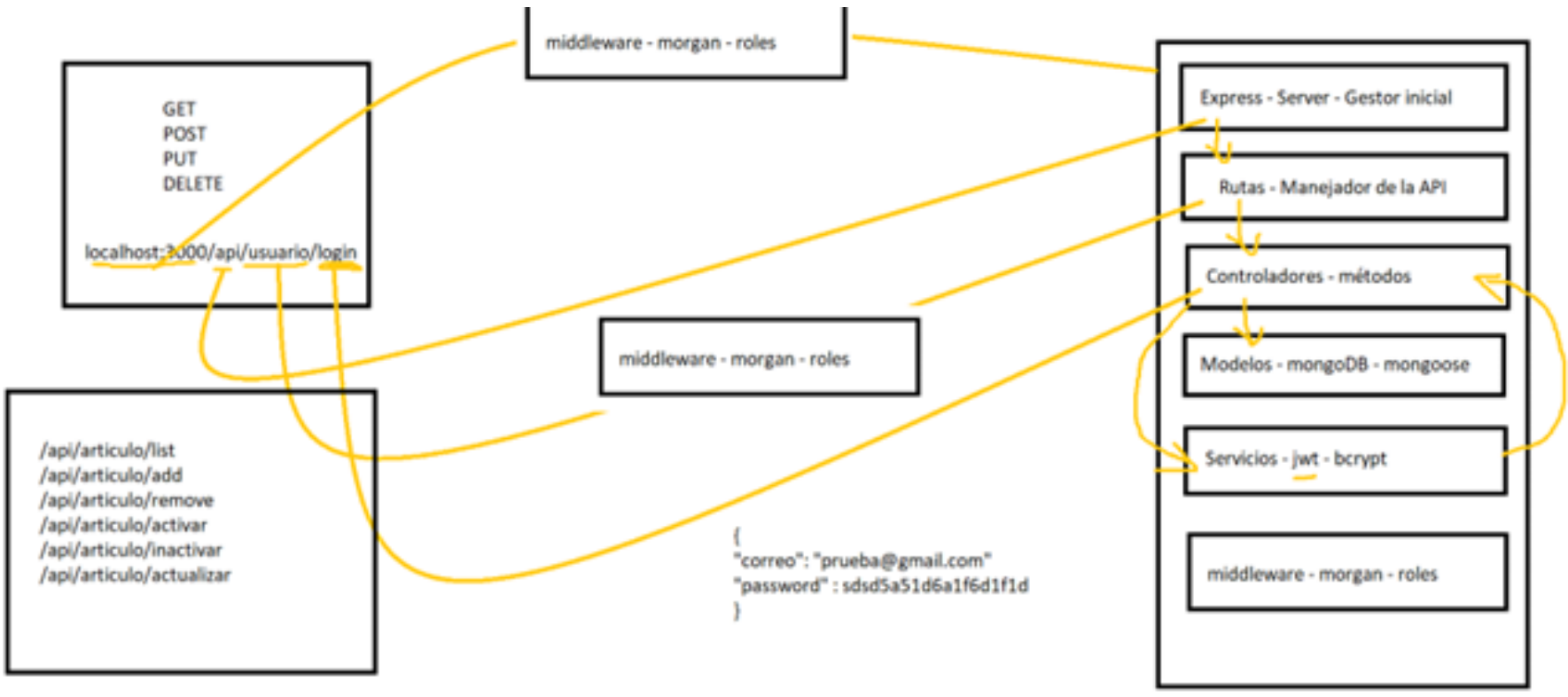
Conociendo la estructura de carpetas de un proyecto, es sencillo describir el recorrido que puede tener una petición, suponiendo que requiere de la activación de todos los elementos mencionados hasta el momento.

Para el ejemplo la petición que se utiliza es la autenticación de un usuario, el usuario debe suministrar un correo y una contraseña para ingresar al sistema, se parte del hecho de no contar con validaciones desde el frontend (tema que se trabaja más adelante), y que toda la responsabilidad recae sobre el backend.

-La petición inicia luego de que el usuario, utilizando un formulario de login, ingresa sus credenciales y da clic al botón ingresar.
-Esta petición viaja, en un método POST, hacía el servidor, la información del usuario se incluye en el body de la petición (nuevamente elementos que se desarrollan más adelante).
-Al llegar al servidor es el entry point, para nuestro caso, el archivo app.js que deberá recibir la petición y validar si coincide con alguna ruta existente.
-Si coincide con una ruta entonces el siguiente paso es enviar la petición al enrutador correspondiente, puede ser que inicialmente se envíe al router principal y sea este quien direccione la petición al router correspondiente.
-Ahora en el router, la petición debe coincidir con alguno de los métodos http que direccione a un controlador, se indicó que esta petición viajaba por un método llamado POST, entonces se debe revisar si existe una ruta que coincida con la ruta de la petición y con el método HTTP.
-Si se encuentra la ruta, entonces la petición pasará por el o los middleware, antes de seguir al controlador, el primer middleware será el del registro de la actividad y el segundo el de la validación de los campos.
-El primer middleware registrará en un archivo de logs el intento de ingreso del usuario, indicando correo y fecha con hora.
-El segundo middleware validará que el campo correo tenga un formato adecuado según corresponda.
-Cuando se superen los middlewares, la petición viajará al controlador, es allí donde se realiza la consulta a los modelos para identificar si el correo del usuario corresponde con un usuario existente.
--En caso afirmativo, el controlador solicita al modelo los datos completos del usuario, para así comprar las contraseñas, que deben estar encriptadas, y verificar si los datos están correctos.

-Si todo es correcto, el controlador generará un token de sesión que permite al usuario validar que es un usuario autenticado, ya que podemos tener middlewares que requieran el token con un perfil determinado para dejar seguir a una ruta o funcionalidad específica.
-El token normalmente almacena algunos datos del usuario como el nombre, correo y rol, y cuenta con una fecha de expiración, todos estos datos, se representan en una cadena de texto encriptada.
-Con el token generado, es hora de responder al usuario y es de nuevo responsabilidad del controlador, enviar los datos, en este caso un mensaje con código exitoso y el token de sesión.
-El controlador utiliza su objeto res como mecanismo para enviar la respuesta y puede llamar a renderizar una vista, por ejemplo la vista del perfil del usuario autenticado, junto con los datos del usuario y su token.
-Finalmente el usuario recibe retroalimentación de su petición, en este caso con el envío a una nueva página donde cargan sus datos de perfil como evidencia de la correcta autenticación.

Cabe aclarar que muchas de las etapas por las que debe pasar la petición, pueden tener flujos alternos, si el usuario no existe, si los datos son incorrectos, si la ruta no está configurada, problemas de conexión con la base de datos, entre otros. Estos flujos alternos pueden generar respuestas tempranas a la petición, siendo no exitosas con mensajes de error o directamente responder con un error del aplicativo.



Revisión detallada del servidor creado con Express

En este punto, se comienza la explicación, por parte del docente, del funcionamiento descrito anteriormente de la petición, sin entrar todavía en el tema de los modelos, el código base se encuentra en express_generator_inicial.zip, también como ejercicio, puede construir el servidor desde 0 e ir instalando dependencias, configurando rutas y demás.

Elementos a tener en cuenta:

-Express
-Morgan
-Cors
-Nodemon
-EJS
-Express Router
-Carpeta Public
-Manejo de rutas 404

De código cabe resaltar dos aspectos, uno es el app.js como entry point y la creación de los enrutadores de la ruta raíz y de usuarios.

Presentar dos peticiones diferentes y como el aplicativo responde de dos maneras diferentes, también tener en cuenta algunos flujos alternos, como rutas no configuradas.

Métodos GET, POST y el use

Dentro del recorrido anterior se encontró que, por ejemplo, en el archivo app.js existen diferentes líneas que implementan el método app.use(), para simplificar un poco, este método le está diciendo a nuestra aplicación dentro de la variable app, que use un recurso o que busque en un recurso determinado, por ejemplo

```
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

Estas dos líneas hacen referencia a un par de routers importados al inicio del archivo

```
let  indexRouter = require('./routes/index');
let  usersRouter = require('./routes/users');
```

Esto quiere decir que para atender las peticiones que lleguen a la raíz, como la página principal de un sitio por ejemplo, se usa al router llamado indexRouter que es importado desde la carpeta routers archivo index.

Si se revisa este archivo aparecen las siguientes líneas de código

```
let express = require('express');
let router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
 res.render('index', { title: 'Express' });
});

module.exports = router;
```

La última línea se encarga de exportar el router para poder ser requerido en otro archivo.

Las primeras dos líneas requieren el router de express para poder hacer uso de sus métodos.

En La línea 5 aparece .get(), esto es similar al .use, solo que acá indica directamente bajo cuál método HTTP responde la ruta en cuestión.

Para entender un poco mejor, se da una breve explicación de los métodos GET y POST

GET

Normalmente utilizado para solicitar información.

Toda la información introducida por el usuario (los llamados “parámetros URL”) se transmiten tan abiertamente como la URL en sí misma. Esto tiene ventajas y desventajas.

Ventajas de GET

Los parámetros URL se pueden guardar junto a la dirección URL como marcador. De esta manera, puedes introducir una búsqueda y más tarde consultarla de nuevo fácilmente. También se puede volver a acceder a la página a través del historial del navegador.

Esto resulta práctico, por ejemplo, si visitas con asiduidad un mismo lugar en Google Maps o si guardas páginas web con configuraciones de filtro determinadas.

Desventajas de GET

La mayor desventaja del método GET es su débil protección de los datos. Los parámetros URL que se envían quedan visibles en la barra de direcciones del navegador y son accesibles sin clave en el historial de navegación, en el caché y en el log de los servidores.

Otra desventaja es que su capacidad es limitada: dependiendo del servidor y del navegador, no es posible introducir más de 2000 caracteres. Además, los parámetros URL solo pueden contener caracteres ASCII (letras, números, signos, etc.) y no datos binarios como archivos de audio o imágenes.

POST

El método POST introduce los parámetros en la solicitud HTTP para el servidor. Por ello, no quedan visibles para el usuario. Además, la capacidad del método POST es ilimitada.

Ventajas de POST

En lo relativo a los datos, como, por ejemplo, al rellenar formularios con nombres de usuario y contraseñas, el método POST ofrece mucha discreción. Los datos no se muestran en el caché ni tampoco en el historial de navegación. La flexibilidad del método POST también resulta muy útil: no solo se pueden enviar textos cortos, sino también otros tipos de información, como fotos o vídeos.

Desventajas de POST

Cuando una página web que contiene un formulario se actualiza (por ejemplo, cuando se retrocede a la página anterior) los datos del formulario deben transferirse de nuevo (puede que alguna vez hayas recibido una de estas advertencias). Por este motivo, existe el riesgo de que los datos se envíen varias veces por error, lo que, en el caso de una tienda online, puede dar lugar a pedidos duplicados. No obstante, las webs modernas de las tiendas suelen estar preparadas para evitar este tipo de problemas.

Además, los datos transferidos con el método POST no pueden guardarse junto al URL como marcador.

Concluyendo

Por ahora, para simplificar un poco el proceso de aprendizaje y posteriormente agregar los métodos HTTP restantes, PUT y DELETE, vamos a utilizar el método GET a la hora de solicitar un recurso en el que no debamos enviar información sensible, como contraseñas, por ejemplo, solicitar una página específica (http://loclahost:3000/articulos) o el detalle del artículo con id 55 (http://loclahost:3000/articulos/55).

Y se utilizará POST cuando necesitemos enviar información sensible o extensa al servidor para obtener una respuesta, por ejemplo, a la hora de autenticarlos en el aplicativo o de llenar un formulario de contacto.