



Aptugo

Código fuente en Aptugo

Guía de estudio

Index

- Introducción3**
- Desarrollo Backend4**
 - ¿Qué es el Desarrollo Backend?4
 - Applications6
 - Builds.....8
 - Models.....10
 - Controllers13
 - Routes15
- Desarrollo Frontend17**
 - ¿Qué es el Desarrollo Frontend?17
 - Components18
 - Pages20
- Conclusiones21**

Introducción

Sabemos que Aptugo nos permite generar aplicaciones completas de manera visual, escribiendo el código fuente por nosotros, pero:

¿Dónde podemos encontrar ese código fuente?

¿Qué es precisamente lo que Aptugo escribe por nosotros?

Conocemos que toda aplicación web consta principalmente de dos capas: backend y frontend.

¿Aptugo diseña ambas o solo una?

¿Dónde podemos encontrar cada una de ellas?

Todas estas preguntas serán respondidas en la presente guía. **Vamos a navegar un poco en los diferentes archivos y carpetas que Aptugo genera por nosotros, todo aquello que deberíamos escribir a mano si programáramos de forma tradicional.** Inspeccionamos los diferentes documentos que se generan, las redes que se constituyen e interacciones entre la interfaz visual de la herramienta y el código.

Es necesario aclarar que **se pretende realizar una aproximación con fines introductorios**, por lo que proponemos la construcción de un ejemplo semejante al propuesto y la exploración individual de las secciones que puedan faltar. Aptugo nos permite generar proyectos descartables, diseñados especialmente con fines experimentales. ¡Animate a explorar!

Desarrollo Backend

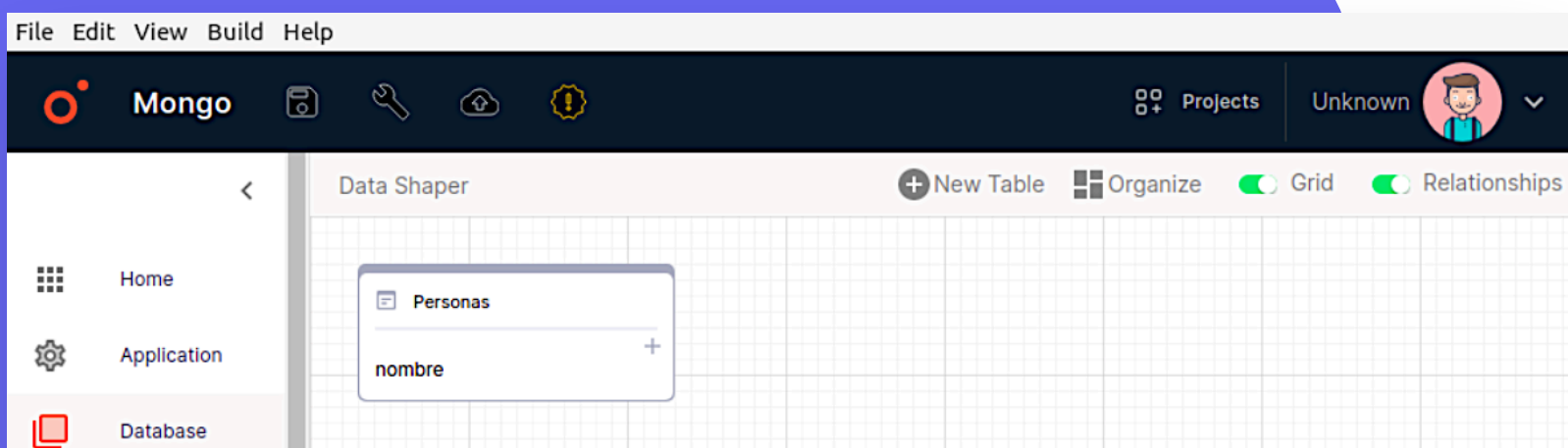
¿Qué es el Desarrollo Backend?

El desarrollo backend es el conjunto de software que sucede detrás de lo que el usuario final puede visualizar desde un navegador web. Está relacionado directamente a las interacciones cercanas a servidores, bases de datos, APIs y todas las estructuras abstractas que permiten la funcionalidad del proyecto, sin interactuar de forma directa con el usuario.

Aptugo ofrece una clara diferenciación entre los distintos tipos de desarrollos vinculados a un proyecto de software, estableciendo secciones determinadas y específicas para los elementos de **Backend y de Frontend**. A continuación, abordaremos qué tipos de carpetas existen, cuales son sus particularidades, y qué tipo de código almacenan.

Es importante destacar que **el código fuente generado por Aptugo a partir de su interfaz visual, no poseerá rastros de la utilización de la herramienta**, es decir, que estaremos analizando un código fuente nativo que incluye las mejores prácticas de las tecnologías actuales en lo referido al desarrollo de software. Por lo tanto, **es equivalente al que escribiría un equipo de programadores con experiencia a mano**, solo que en vez de hacerlo de forma manual demorando días o semanas, lo haría en minutos y de manera visual.

Para ilustrar los conceptos, esta guía contendrá un proyecto denominado “Mongo” que inicialmente está compuesto por una tabla denominada “Personas” y que, a su vez, contiene un campo denominado “nombre”.



A lo largo del desarrollo de esta guía de estudio, se estarán incorporando modificaciones desde la interfaz visual de Aptugo para realizar la asociación de ciertos casos puntuales con el código fuente generado.

Para poder visualizar el código fuente, debemos remitirnos a las carpetas donde éste se guarda.

En un principio, podemos establecer:

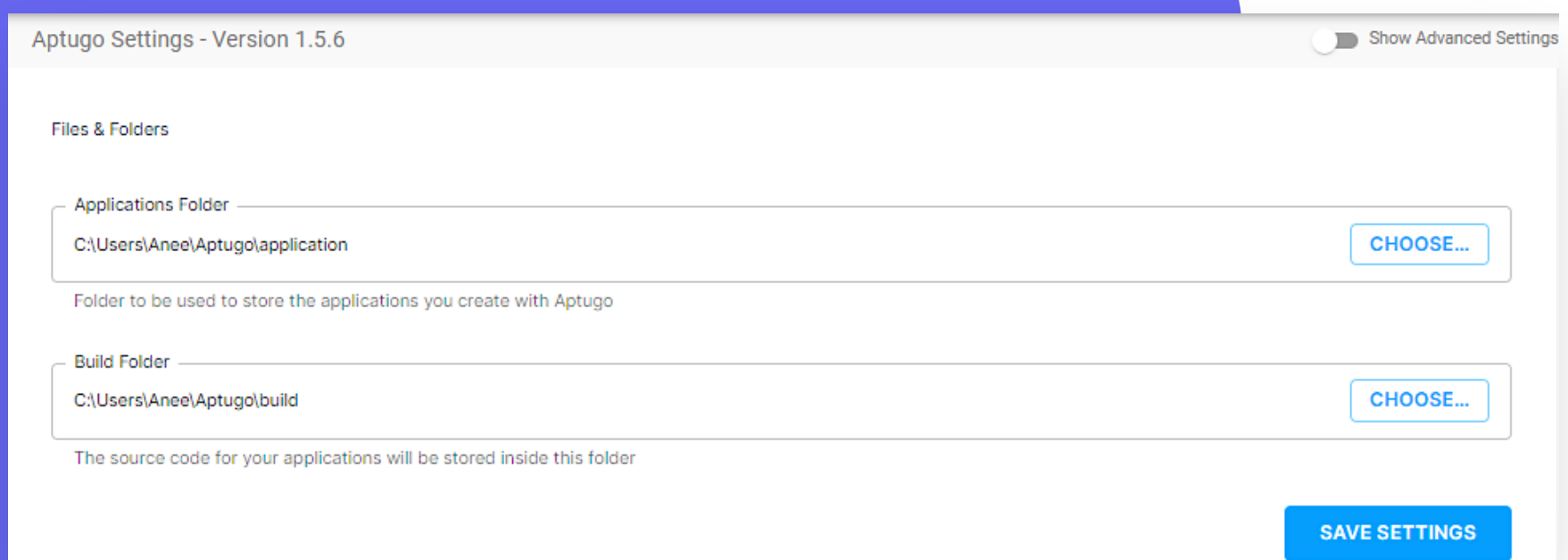
- **Applications:**

```
\Users\NombredeUsuario\Aptugo\application\Mongo
```

- **Builds:**

```
\Users\NombredeUsuario\Aptugo\build\Mongo
```

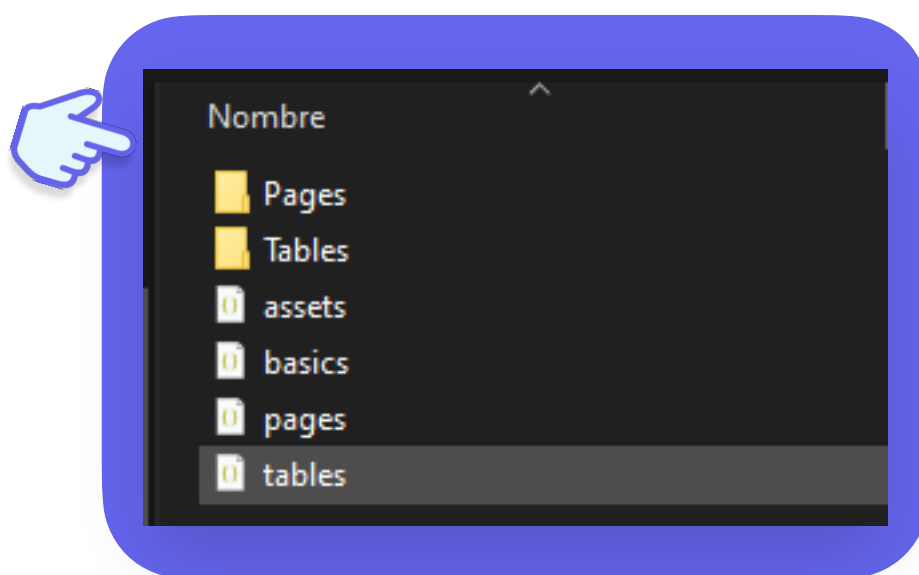
Es importante destacar que **las rutas pueden variar según la configuración que el usuario haya realizado**, por lo que podemos verificarlas directamente desde Aptugo, en la sección “**settings**”:



Applications

La carpeta **Applications** es la que contiene las carpetas y los archivos que conforman la representación visual, parametrización y opciones utilizadas en la construcción realizada con Aptugo. **A partir de esta carpeta es que, al realizar las opciones de “Save” y “Build your project”, se generará su correspondiente carpeta dentro del directorio builds** donde tendremos el código fuente deseado.

Todo proyecto generado con Aptugo presenta, en la carpeta Applications, los siguientes contenidos:

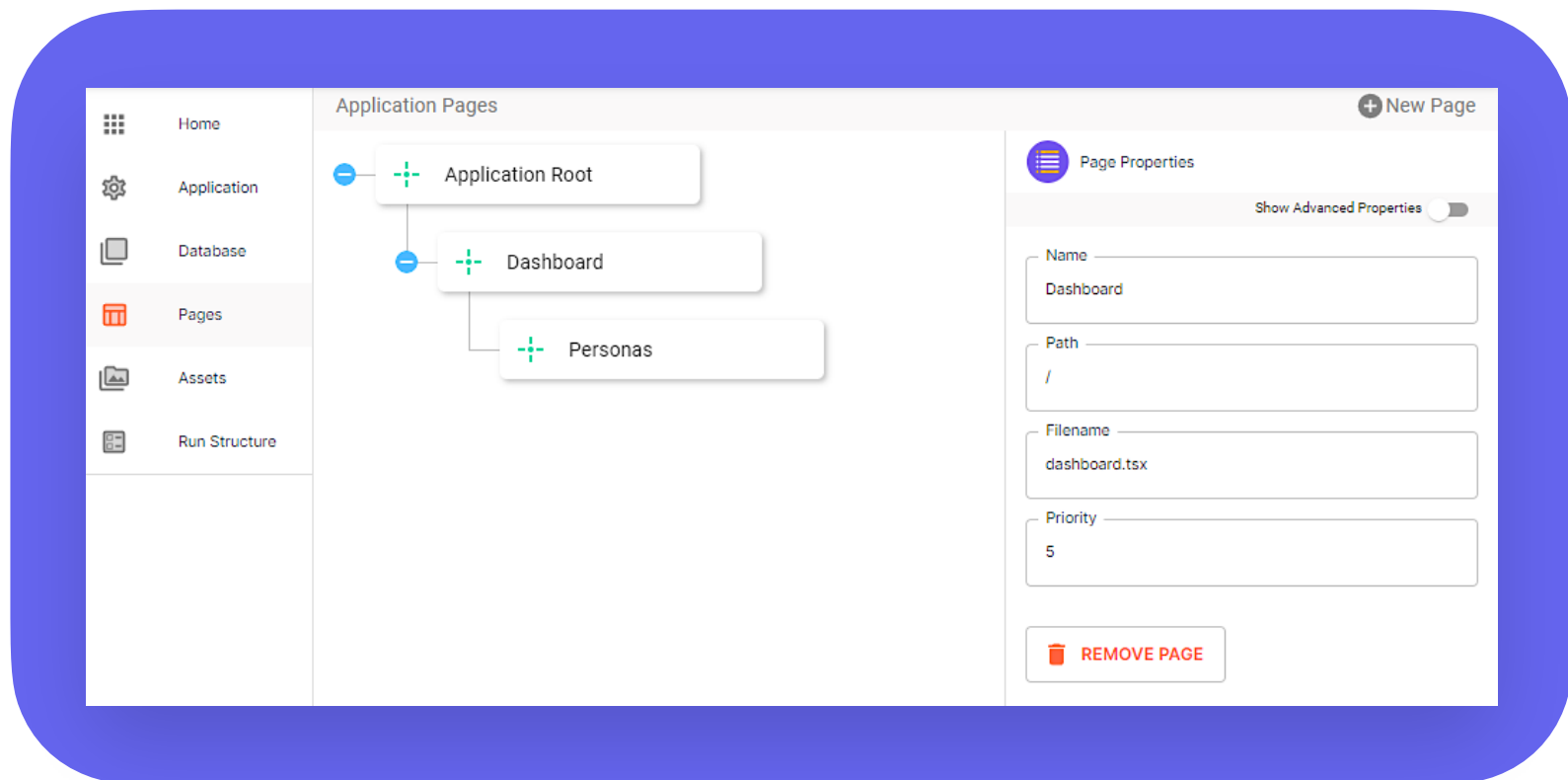


Dentro de la carpeta Pages pueden verse diferentes archivos de formato .json, que responden a elementos y páginas que Aptugo va construyendo por defecto para el correcto funcionamiento de la interfaz de la aplicación.

En el ejemplo que sigue, se presenta uno de los archivos de dicha carpeta, el cual contiene las opciones que figuran como propiedad de la página Dashboard:

```
{ } a0ZlK80s.json X
Pages > { } a0ZlK80s.json > ...
1  {
2    "unique_id": "a0ZlK80s",
3    "name": "Dashboard",
4    "type": "page",
5    "path": "/",
6    "filename": "dashboard.tsx",
7    "prevent_delete": false,
8    "collapseStatus": "expand",
9    "parent": "K5PgPazk"
10 }
```

Y su representación desde Aptugo:



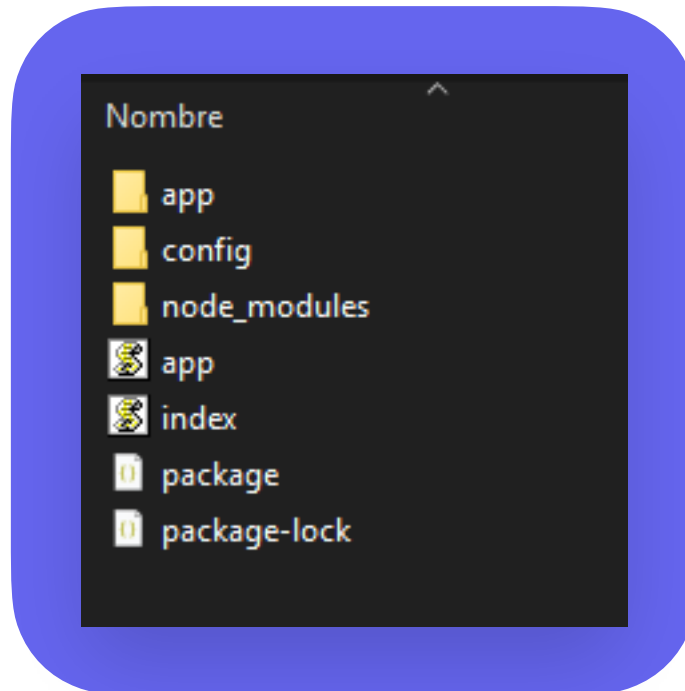
A fines didácticos, y por cuestiones de alcance del presente curso, sólo abordaremos lo anteriormente mencionado. Se recomienda que el lector se interiorice en los mismos mediante navegación individual.

Builds

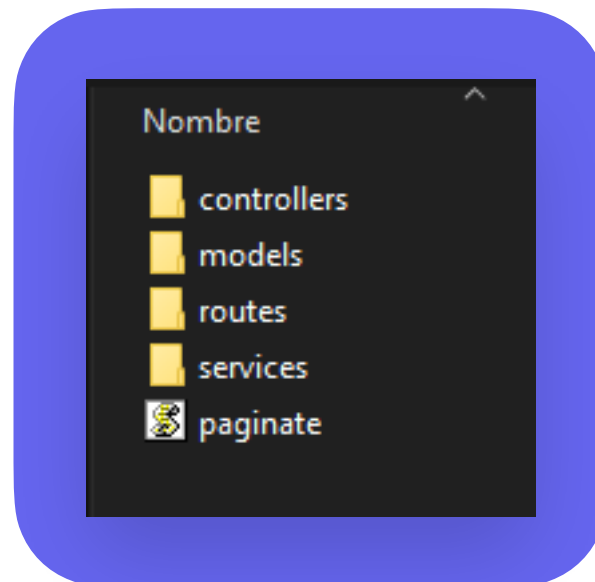
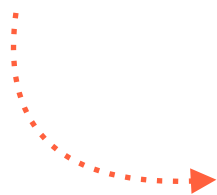
La carpeta **builds** es la que contiene las carpetas y los archivos con el código fuente generado por Aptugo.

En la presente sección, como abordamos la capa **Backend** del proyecto, nos enfocaremos en la carpeta de nombre correspondiente: Back-end

Los contenidos iniciales de la misma son los siguientes:




El siguiente paso consiste en **visualizar** los contenidos dentro de la **carpeta app**:



Recordemos que se deben realizar **“Save”** y **“Build your project”** para **visualizar el proyecto en builds**.


Existe un patrón de diseño de software denominado MVC (Model-View-Controller) que tiene por objetivo separar la construcción del mismo en 3 capas:

Model: Relacionado al modelo de datos implementado. Se corresponde con la carpeta del mismo nombre, cuyo contenido es:

Nombre	Fecha de modificación	Tipo
 personas.model	15/08/2022 13:04	Archivo JavaScript

View: Relacionado a la visualización de los elementos a través del navegador como la interfaz del usuario. Se verá reflejado en los contenidos a abordar en la siguiente sección: Desarrollo Frontend

Controller: Relacionado a la lógica de las operaciones del software. Se corresponde con la carpeta del mismo nombre, cuyo contenido es:

Nombre	Fecha de modificación	Tipo
 personas.controller	15/08/2022 13:04	Archivo JavaScript

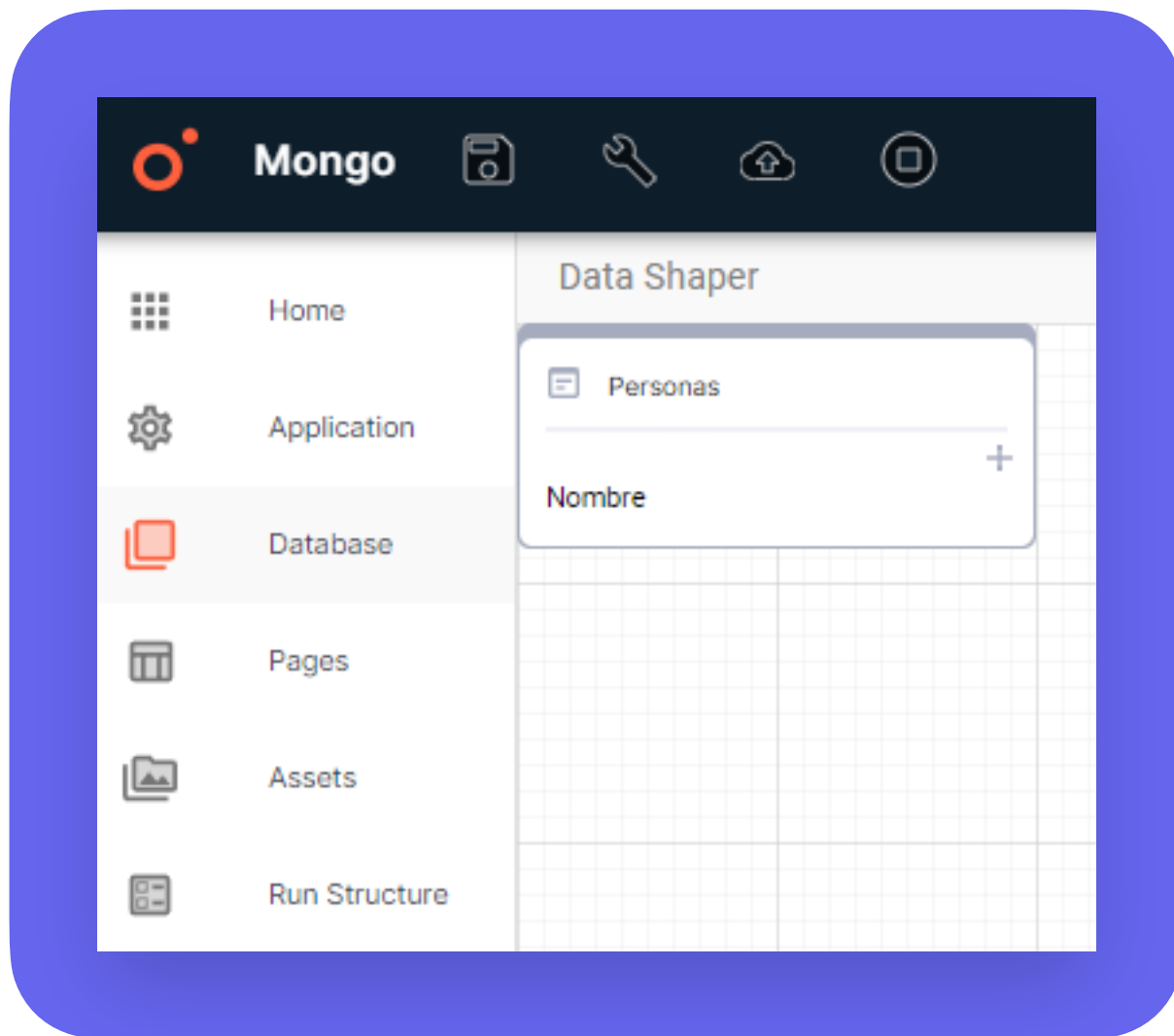
A continuación profundizaremos en las secciones más importantes.

Models

Hace referencia al modelo de datos de Aptugo, y su representación a través del código fuente generado.

Para una determinada tabla, existirá su correspondiente dentro de:

- [back-end/app/models/tabla.model.js](#)

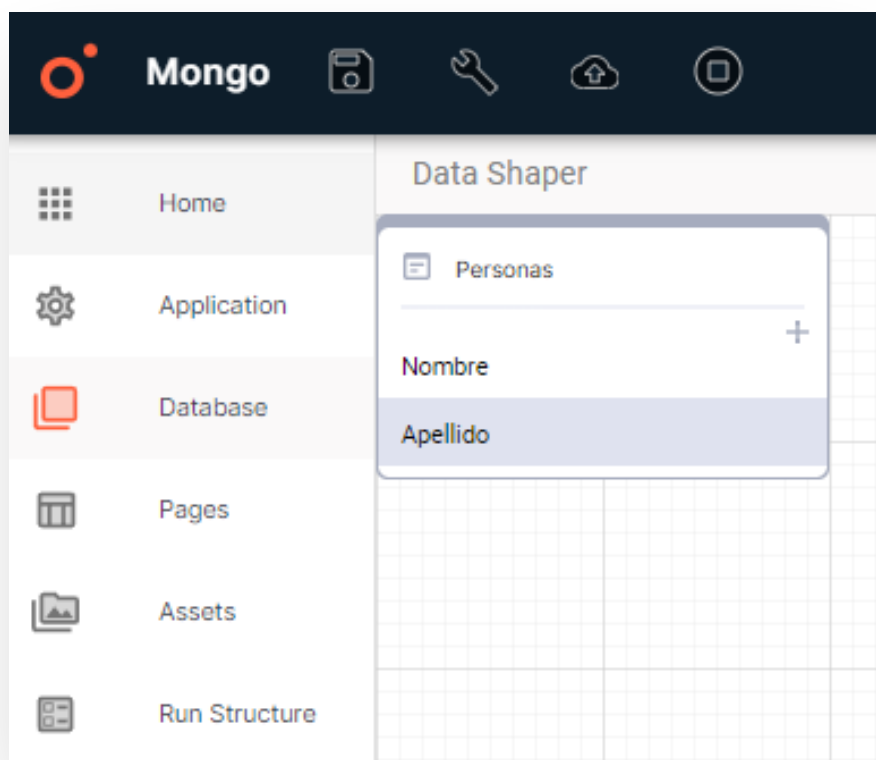


En nuestro ejemplo, consideremos que el proyecto Mongo posee una tabla llamada **Personas**, por lo que tendremos el archivo será:

- [back-end/app/models/personas.model.js](#)

```
JS personas.model.js X
back-end > app > models > JS personas.model.js > ...
1  const mongoose = require('mongoose')
2  const mongoosePaginate = require('mongoose-paginate-v2')
3
4  const PersonSchema = mongoose.Schema(
5  {
6  Nombre: {
7    type: String,
8  },
9  },
10 {
11   timestamps: true,
12   toJSON: { virtuals: true },
13 }
14 )
15
16 PersonSchema.plugin(mongoosePaginate)
17 PersonSchema.index({
18   Nombre: 'text',
19 })
20
21 const myModel = (module.exports = mongoose.model('Personas', PersonSchema, 'personas'))
22 myModel.schema = PersonSchema
23
```

Probemos agregar un nuevo campo denominado **Apellido** desde Aptugo, para evaluar los cambios que se generan en el código fuente.



Recordemos, luego de generar dicho campo, **hacer “save” and “build your project”** para que los mismos se implementen en el código fuente.

JS personas.model.js X

back-end > app > models > JS personas.model.js > ...

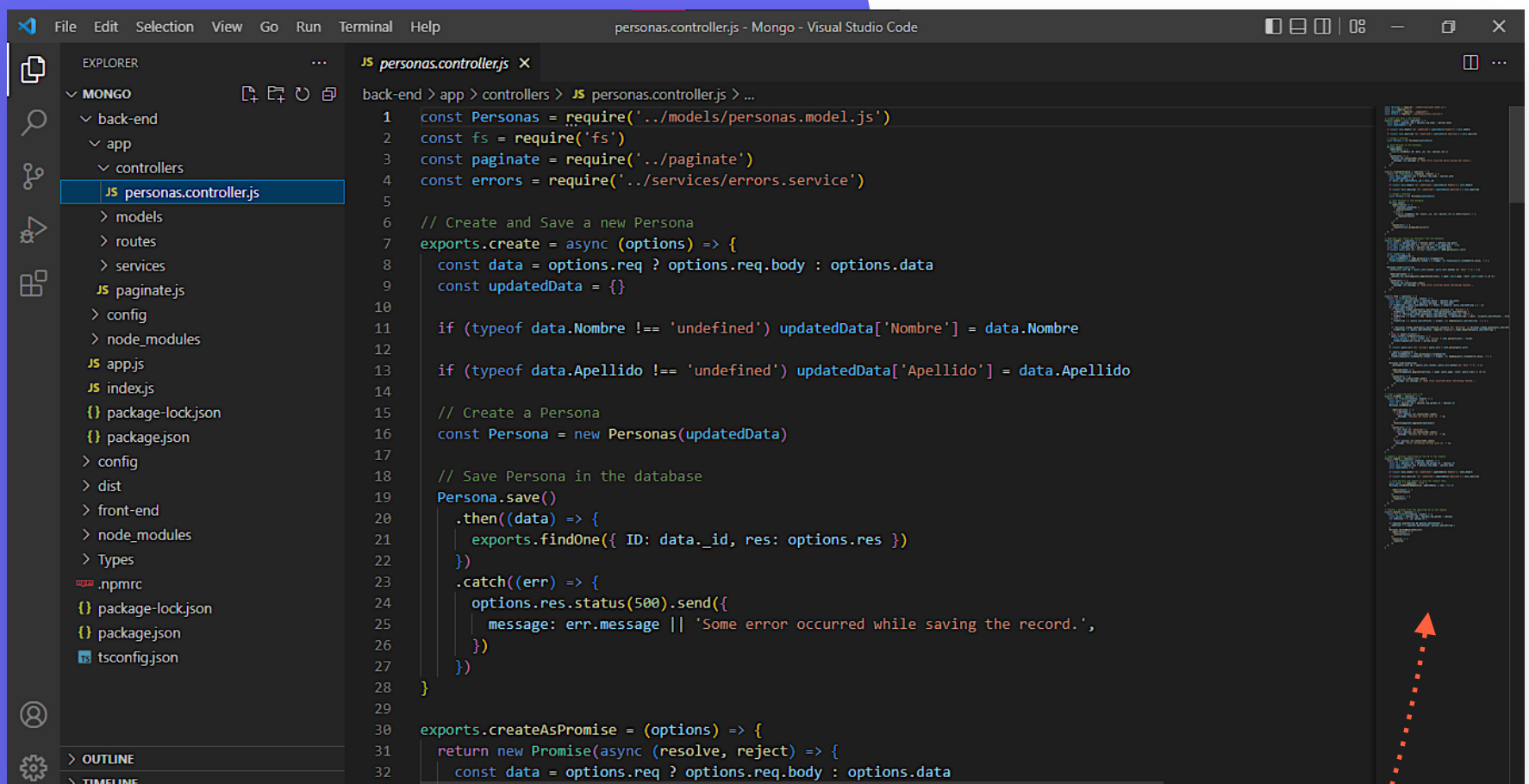
```
1  const mongoose = require('mongoose')
2  const mongoosePaginate = require('mongoose-paginate-v2')
3
4  const PersonSchema = mongoose.Schema(
5    {
6      Nombre: {
7        type: String,
8      },
9      Apellido: {
10       type: String,
11     },
12   },
13   {
14     timestamps: true,
15     toJSON: { virtuals: true },
16   }
17 )
18
19 PersonSchema.plugin(mongoosePaginate)
20 PersonSchema.index({
21   Nombre: 'text',
22   Apellido: 'text',
23 })
24
25 const myModel = (module.exports = mongoose.model('Personas', PersonSchema, 'personas'))
26 myModel.schema = PersonSchema
27
```

Controllers

Hace referencia a la lógica de las operaciones del software tales como Crear, Guardar, Buscar, Actualizar y Borrar.

De igual forma como sucedía en models, **también existirá en dentro de controllers el archivo correspondiente:**

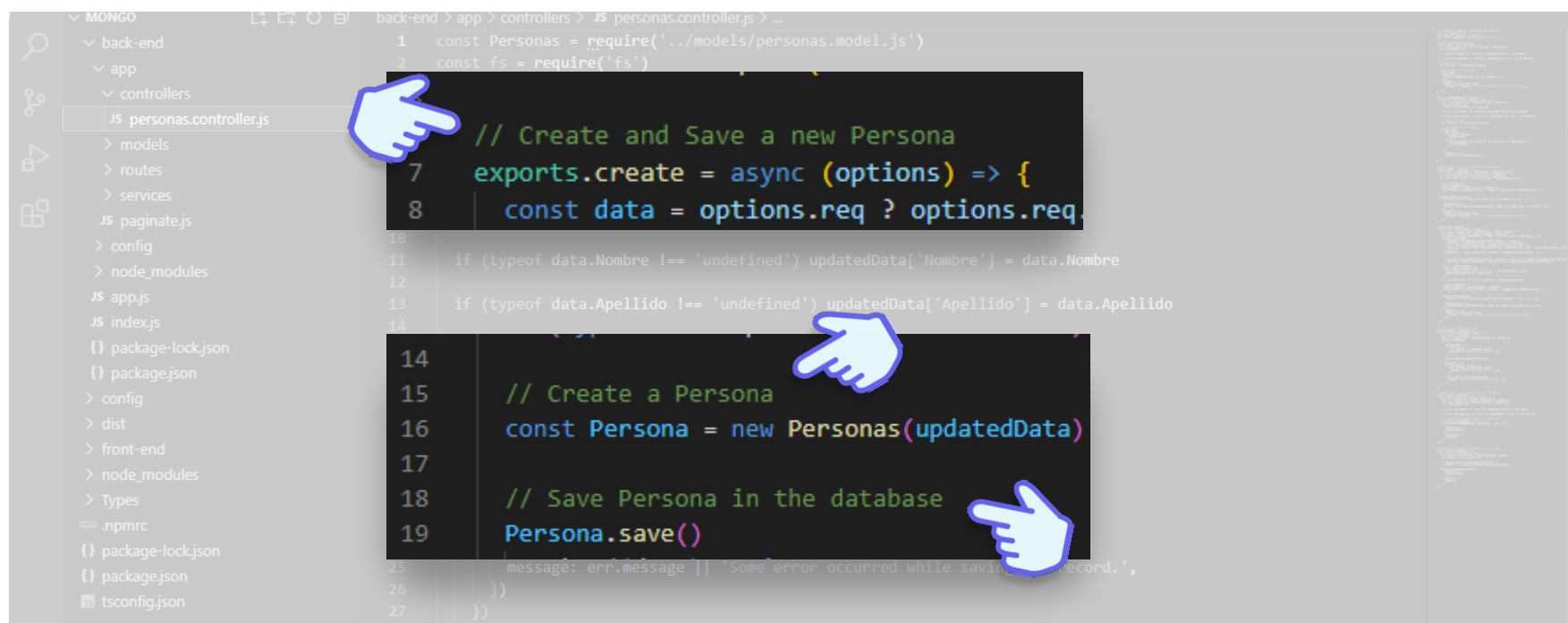
- [back-end/app/controllers/personas.controller.js](#)



```
1 const Personas = require('../models/personas.model.js')
2 const fs = require('fs')
3 const paginate = require('../paginate')
4 const errors = require('../services/errors.service')
5
6 // Create and Save a new Persona
7 exports.create = async (options) => {
8   const data = options.req ? options.req.body : options.data
9   const updatedData = {}
10
11   if (typeof data.Nombre !== 'undefined') updatedData['Nombre'] = data.Nombre
12
13   if (typeof data.Apellido !== 'undefined') updatedData['Apellido'] = data.Apellido
14
15   // Create a Persona
16   const Persona = new Personas(updatedData)
17
18   // Save Persona in the database
19   Persona.save()
20     .then((data) => {
21       exports.findOne({ ID: data._id, res: options.res })
22     })
23     .catch((err) => {
24       options.res.status(500).send({
25         message: err.message || 'Some error occurred while saving the record.',
26       })
27     })
28 }
29
30 exports.createAsPromise = (options) => {
31   return new Promise(async (resolve, reject) => {
32     const data = options.req ? options.req.body : options.data
```



A la derecha se ubica una miniatura correspondiente a la estructura del código fuente, ¡imagina si debiéramos escribir todo eso a mano!



Los **comentarios** incluidos en el código fuente son de utilidad para comprender la función que realiza cada bloque de código en particular:

// Create and Save a new Persona

// Create a Persona

// Save Persona in the database

// Retrieve and return all Personas from the database

// Find a single Persona with a ID

// Update a persona identified by the ID in the request

// Delete a persona with the specified ID in the request



La lógica se repetirá para todos y cada uno de los modelos contruidos, y **Aptugo** se encargará de **generar el código fuente**, a fin de **evitar la escritura repetitiva** de las mismas operaciones para cada una de las nuevas tablas creadas.

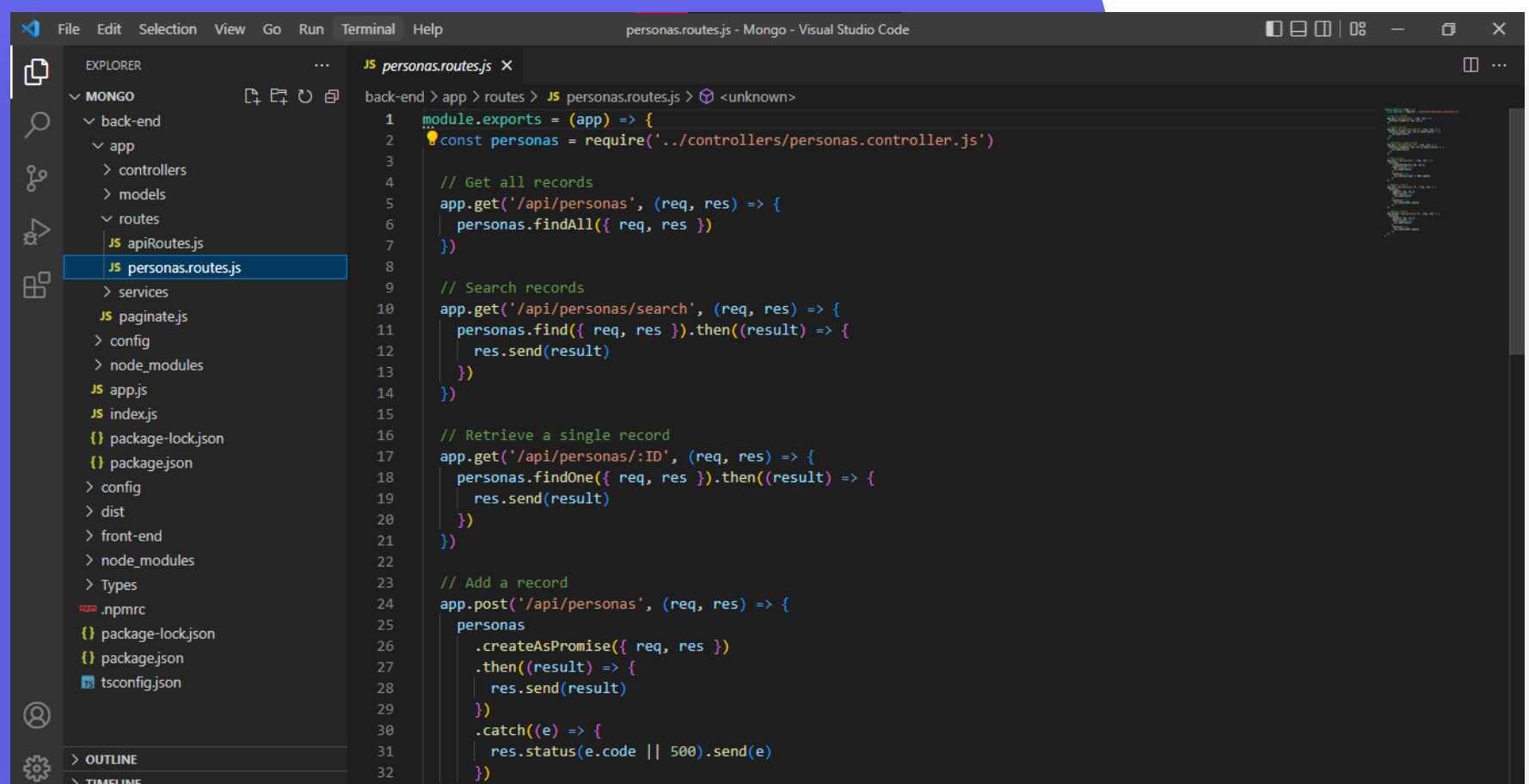
Routes

Tal como su nombre lo indica, de forma análoga a la opción que Aptugo ofrece, será el código fuente que resolverá todas las rutas a partir de las cuales la API recibirá peticiones (requests) y ofrecerá respuestas (responses).

En el código fuente, dentro de la carpeta Builds→Back-end→app→Routes podemos encontrar el código fuente dentro del archivo:

- [back-end/app/routes/personas.routes.js](#)

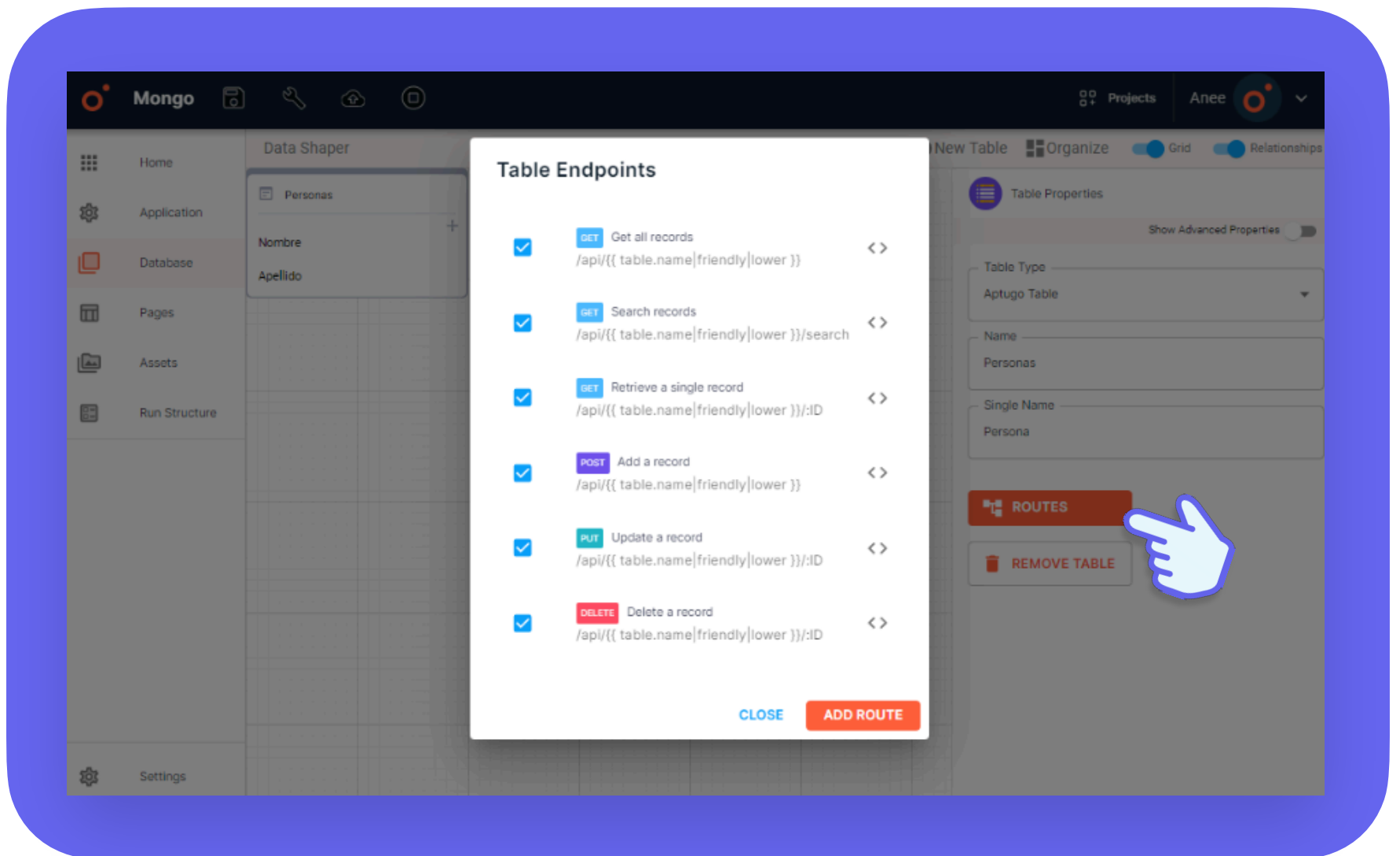
Lo podemos visualizar de la siguiente forma:



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure under the 'MONGO' folder, with 'personas.routes.js' selected under the 'routes' directory. The main editor window shows the code for 'personas.routes.js', which defines several REST API endpoints for a 'personas' resource.

```
1 module.exports = (app) => {
2   const personas = require('../controllers/personas.controller.js')
3
4   // Get all records
5   app.get('/api/personas', (req, res) => {
6     personas.findAll({ req, res })
7   })
8
9   // Search records
10  app.get('/api/personas/search', (req, res) => {
11    personas.find({ req, res }).then((result) => {
12      res.send(result)
13    })
14  })
15
16  // Retrieve a single record
17  app.get('/api/personas/:ID', (req, res) => {
18    personas.findOne({ req, res }).then((result) => {
19      res.send(result)
20    })
21  })
22
23  // Add a record
24  app.post('/api/personas', (req, res) => {
25    personas
26      .createAsPromise({ req, res })
27      .then((result) => {
28        res.send(result)
29      })
30      .catch((e) => {
31        res.status(e.code || 500).send(e)
32      })
33  })
34 }
```

Además, **las mismas rutas pueden ubicarse dentro de la plataforma de Aptugo, al seleccionar la opción “Routes” de tabla pretendida, revelando las **rutas** con sus parámetros:**



Hasta aquí las secciones más importantes del desarrollo Backend. Retomando comentarios anteriores, se pretende abordar cada sección de forma práctica e introductoria. **Para ahondar más en el tema, se propone experimentar e indagar en cada sección no tratada, realizar cambios y comprender la lógica de construcción de Aptugo.**

Desarrollo Frontend

¿Qué es el Desarrollo Frontend?

El desarrollo frontend es el conjunto de software que representa la interacción del **usuario final con los elementos visuales** a los que accede desde un navegador web.

Está relacionado al **aspecto visual, desde su diseño hasta la maquetación, por medio de imágenes, botones, tipos de letra, y otros recursos gráficos**. Las bases del desarrollo frontend están dadas por HTML y CSS mientras que actualmente se cuentan con múltiples librerías frontend que complementan y facilitan la utilización de distintos elementos.

En esta sección, **nos enfocaremos en los contenidos de la carpeta Front-end ubicada dentro de la carpeta Builds** puesto que, tal como su nombre indica, nuclea las carpetas y el código fuente asociado al desarrollo Front-end del proyecto de software.

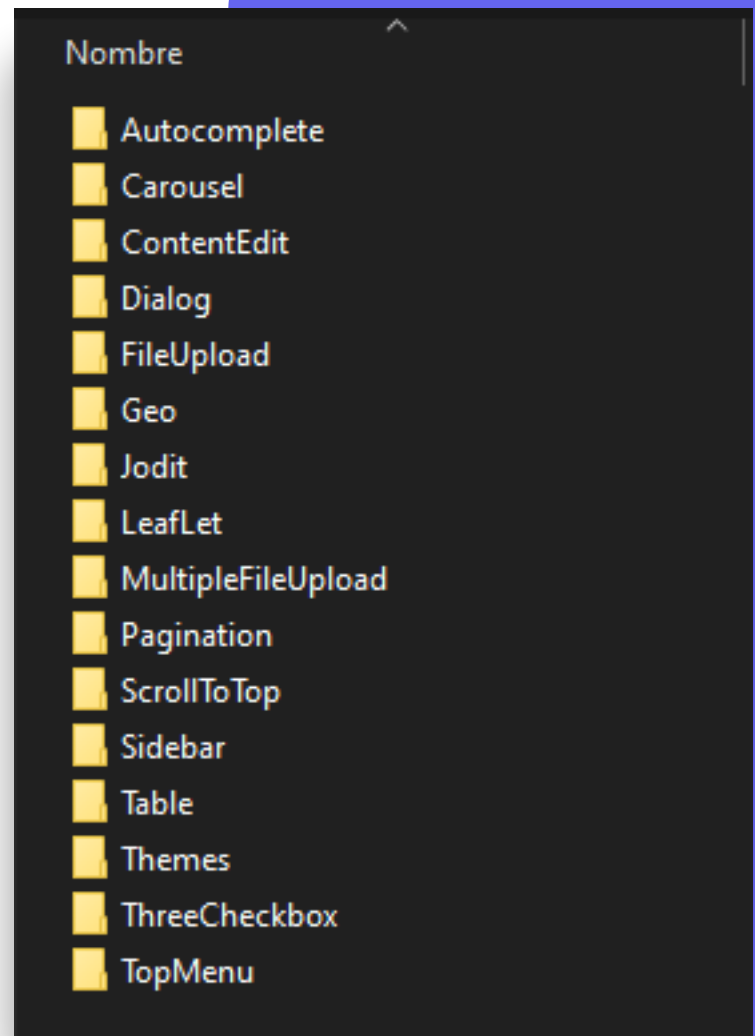


A continuación, **desarrollaremos los directorios más importantes**.

Components

El siguiente paso consiste en **visualizar los contenidos dentro de la carpeta `components`**, el cual posee diferentes archivos de extensión `.txt` y `.sass`, con configuración predeterminada para el correcto funcionamiento de ciertos elementos.

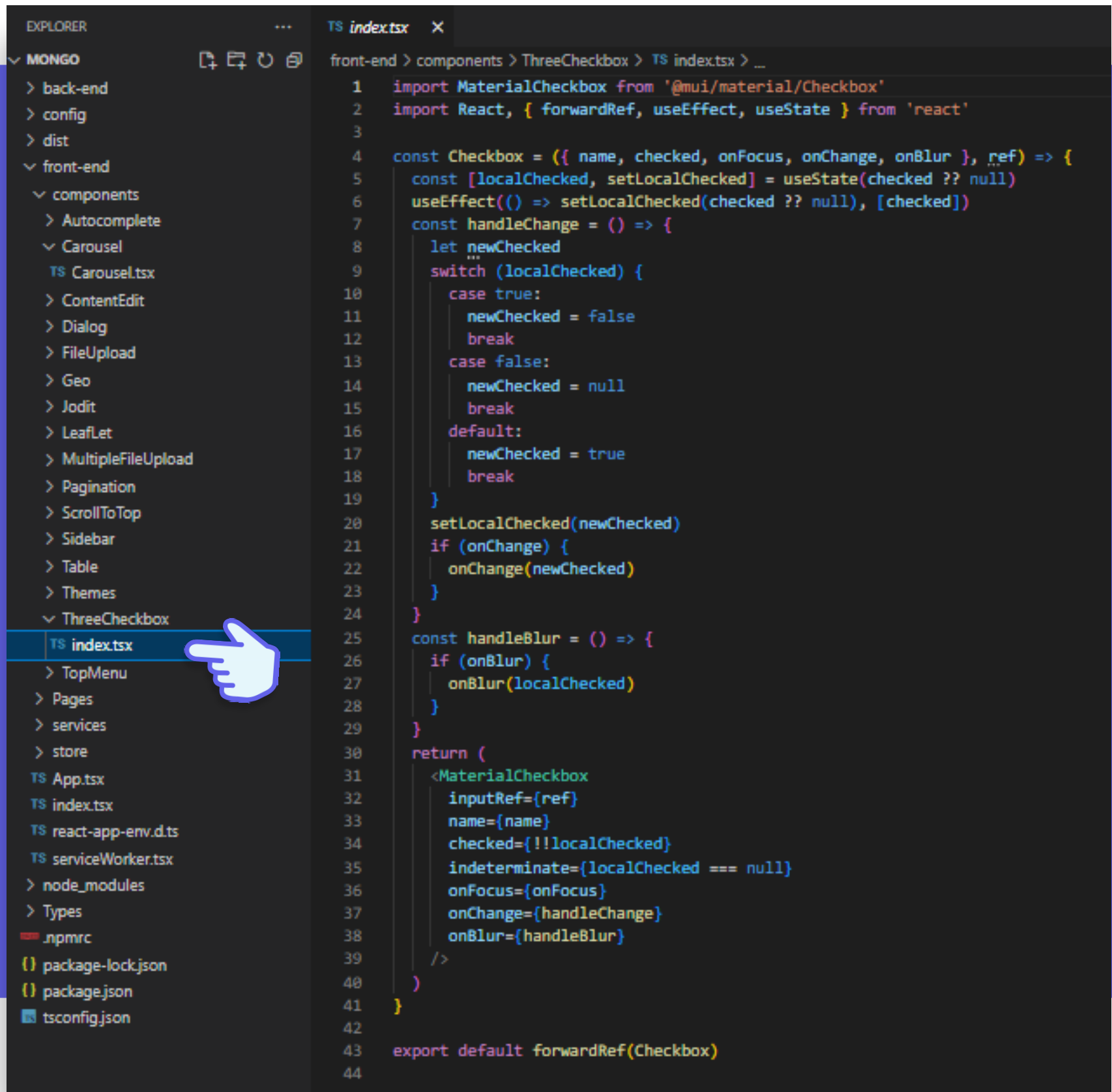
La misma será de utilidad en caso de implementarlos en nuestro proyecto:



Pueden observarse distintos elementos ya trabajados anteriormente como:

- **Dialog**
- **Leaflet**
- **Sidebar**
- **Table**

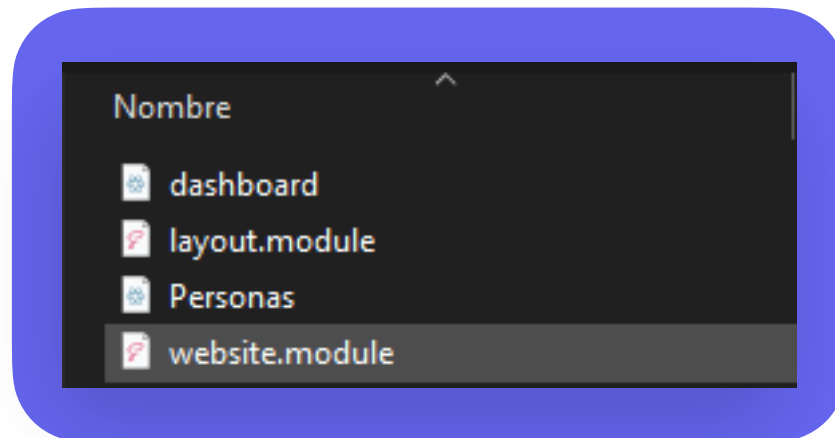
A modo de ejemplo, **podemos abrir el archivo `.txt` del elemento `ThreeCheckbox` para ver su contenido:**



```
1 import MaterialCheckbox from '@mui/material/Checkbox'
2 import React, { forwardRef, useEffect, useState } from 'react'
3
4 const Checkbox = ({ name, checked, onFocus, onChange, onBlur }, ref) => {
5   const [localChecked, setLocalChecked] = useState(checked ?? null)
6   useEffect(() => setLocalChecked(checked ?? null), [checked])
7   const handleChange = () => {
8     let newChecked
9     switch (localChecked) {
10      case true:
11        newChecked = false
12        break
13      case false:
14        newChecked = null
15        break
16      default:
17        newChecked = true
18        break
19    }
20    setLocalChecked(newChecked)
21    if (onChange) {
22      onChange(newChecked)
23    }
24  }
25   const handleBlur = () => {
26     if (onBlur) {
27       onBlur(localChecked)
28     }
29   }
30   return (
31     <MaterialCheckbox
32       inputRef={ref}
33       name={name}
34       checked={!localChecked}
35       indeterminate={localChecked === null}
36       onFocus={onFocus}
37       onChange={handleChange}
38       onBlur={handleBlur}
39     />
40   )
41 }
42
43 export default forwardRef(Checkbox)
44
```

Pages

Hace referencia al **código fuente de la representación visual creada en Aptugo desde la sección [Pages](#)**.



Los contenidos de la carpeta coinciden nuevamente con las extensiones [.tsx](#) y [.scss](#).

Los **archivos .scss** creados por el usuario se encontrarán en la sección [Builds→Dist→css](#), mientras que **layout.module.scss** y **website.module.scss** los crea Aptugo por defecto.

Con fines orientativos, **describamos brevemente ambas extensiones:**

[.tsx](#)

Los archivos con extensión **.tsx hacen referencia a TypeScript**, una extensión de JavaScript que permite desarrollar interfaces y combinarlas con componentes React.

La utilización de etiquetas similares a HTML, junto a la utilización de lógica de programación hacen de TypeScript un potente lenguaje de programación.

[.scss](#)

Los archivos con extensión **.scss hacen referencia a Syntactically Awesome Style Sheets**, una extensión de CSS (Cascading Style Sheets) que permite escribir hojas de estilo con funcionalidades avanzadas.

Conclusiones

Esta **guía de estudio** brinda los conceptos elementales del Desarrollo Backend y Frontend de una forma práctica y los **relaciona con su lógica de construcción en Aptugo, reflejando el vínculo entre la parte visual y el código fuente**.

La implementación de forma gradual de nuevos campos y elementos permitirá comprender cuál es el código fuente que Aptugo genera para cada nueva incorporación.



Se sugiere ir avanzando progresivamente, agregando recursos de ambas capas cada vez más complejos, para verificar el impacto de la construcción visual convertida a código fuente.

El estudio minucioso de las carpetas back-end y front-end profundizará el conocimiento y permitirá reconocer nuevas porciones de código fuente que se genera y ejecuta para cada proyecto nuevo, acrescentando no solo la comprensión respecto al funcionamiento de la herramienta, sino también a la configuración y escritura del código fuente.