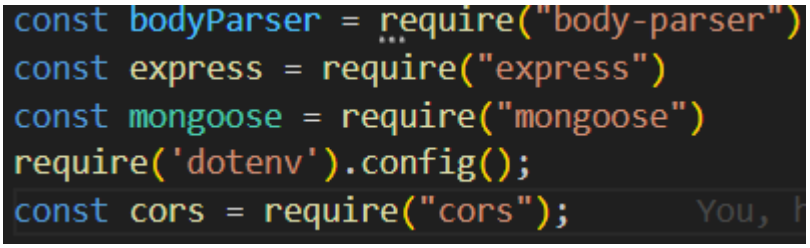
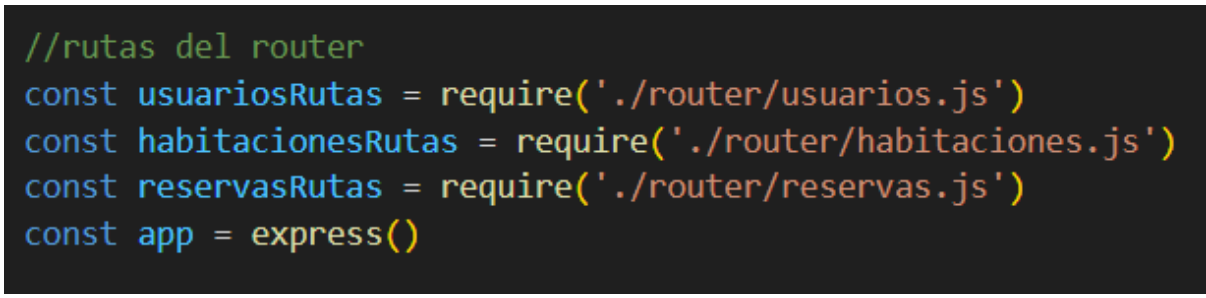
Tecnologías Principales

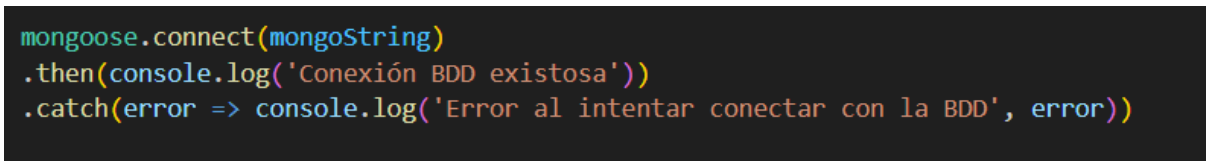
- **Node.js** y **Express.js** para la creación del servidor y la definición de endpoints (rutas).
- **MongoDB** con **Mongoose** para la definición de esquemas y modelos de datos (Habitaciones, Tipos de Habitación, Usuarios, Reservas, etc.).
- **dotenv** para manejar variables de entorno (como la URL de la base de datos y el puerto).
- **Multer** para la subida y administración de archivos (imágenes de habitaciones).
- **CORS** para permitir peticiones desde distintos dominios.



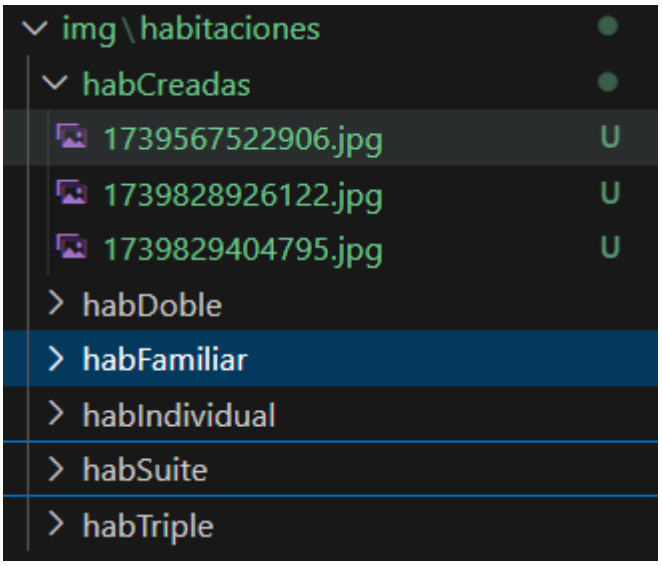
Estructura de Carpetas y Configuración Principal

- Archivo principal **app.js** donde se:
  - Configuran los middlewares (**bodyParser**, **cors**, etc.).
  - 

  - Se importan y utilizan las rutas (“routers”) para las diferentes funcionalidades.
  - 

  - Se conecta a la base de datos MongoDB usando la URL definida en las variables de entorno.
  - 

- Rutas principales importadas:
  - **usuarios.js** → Manejo de usuarios.
  - **habitaciones.js** → Manejo de habitaciones.
  - **reservas.js** → Manejo de reservas.

- Directorio de imágenes (**img**) que aloja las fotos subidas de las habitaciones.
- 

- Configuración de **multer** para especificar dónde se guardan las imágenes y con qué nombre.
- ```
  const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, "img/habitaciones/habCreadas"); // Carpeta donde se guardan las imágenes
    },
    filename: (req, file, cb) => {
        cb(null, Date.now() + path.extname(file.originalname)); // Nombre único
    }
  });

  const upload = multer({ storage: storage });
  ```

Endpoints y Funcionalidades para "Habitaciones" (definidos en **habitaciones.js**)

- **GET /api/habitacion**
  Retorna todas las habitaciones. Permite realizar filtrado avanzado mediante query params:
  - **nombreTipo**: filtrar por tipo de habitación.
  - **precioMin, precioMax**: rango de precios.
  - **cantHuespedes**: filtrar por capacidad de huéspedes.
  - **disponible**: filtrar por disponibilidad (true/false).
  - **pisoMin, pisoMax**: filtrar por rango de pisos.
  ```

```
let { nombreTipo, precioMin, precioMax, cantHuespedes, disponible, pisoMin, pisoMax } = req.query;
let filtro = {};

// Filtrar por tipo de habitación
if (nombreTipo) filtro["tipoHabitacion.nombreTipo"] = nombreTipo;

// Filtrar por precio mínimo y máximo
if (precioMin) filtro.precio = { $gte: parseFloat(precioMin) };
if (precioMax) filtro.precio = { ...filtro.precio, $lte: parseFloat(precioMax) };

// Filtrar por cantidad de huéspedes (adultos + menores)
if (cantHuespedes) {
    cantHuespedes = parseInt(cantHuespedes);
    filtro.$expr = {
        $gte: [
            { $add: ["$tipoHabitacion.capacidadPersonas.adultos", "$tipoHabitacion.capacidadPersonas.menores"] },
            cantHuespedes
        ]
    };
}       You, hace 5 días • añadido filtro e img
// Filtrar por disponibilidad
if (disponible !== undefined) filtro.disponible = disponible === "true";

// Filtrar por rango de pisos
if (pisoMin) filtro.piso = { $gte: parseInt(pisoMin) };
if (pisoMax) filtro.piso = { ...filtro.piso, $lte: parseInt(pisoMax) };

  const habitaciones = await Habitacion.find(filtro)
```

- **GET /api/habitacion/:id**
  Devuelve los datos de una habitación específica, buscándola por su ID en MongoDB.

```
//devuelve la habitacion del id en los parámetros
router.get("/habitacion/:id", async (req, res) =>{
    try{
        const habitacion = await Habitacion.findById(req.params.id)
        res.status(200).json(habitacion)

    }catch(error){
        res.status(400).json({message: 'Error al obtener habitacion', error})
    }
})
```

- **POST /api/habitacion** (con subida de imagen)
  Crea una nueva habitación a partir de los datos enviados en el body (JSON) y sube la imagen correspondiente con **multer**, almacenando la ruta en la base de datos.

```
// Convertir el JSON recibido a objeto JavaScript
let habitacionData = JSON.parse(req.body.habitacion);
      You, hace 1 segundo • Uncommitted changes
// Agregar las rutas de las imágenes al objeto antes de guardarlo
let nuevaHabitacion = new Habitacion({
  numeroHabitacion: habitacionData.numeroHabitacion,
  tipoHabitacion: {
      nombreTipo: habitacionData.tipoHabitacion.nombreTipo,
      precioBase: habitacionData.tipoHabitacion.precioBase,
      capacidadPersonas: {
          adultos: habitacionData.tipoHabitacion.capacidadPersonas.adultos,
          menores: habitacionData.tipoHabitacion.capacidadPersonas.menores
      },
      capacidadCamas: habitacionData.tipoHabitacion.capacidadCamas
  },
  descripcion: habitacionData.descripcion,
  precio: habitacionData.precio,
  fotos: [req.file.path], // Aquí almacenamos las rutas de las imágenes subidas
  camas: {
      individual: habitacionData.camas.individual,
      doble: habitacionData.camas.doble
  },
  dimensiones: habitacionData.dimesiones,
  disponible: habitacionData.disponible,
  piso: habitacionData.piso
);
// Guardar en MongoDB
await nuevaHabitacion.save();
```

- **DELETE /api/habitacion/:numero**
  Elimina la habitación cuyo **numeroHabitacion** coincida con el parámetro de la ruta.

```
//elimina la habitación especificada en los parámetros
router.delete("/habitacion/:numero", async (req, res) =>{
    try {
        const habitacion = await Habitacion.deleteOne(
          {numeroHabitacion: req.params.numero}
        )
        // Verificar si se eliminó correctamente
        if (resultado.deletedCount > 0) {
          res.json({ message: "Habitación eliminada correctamente" });
      } else {
          res.status(404).json({ message: "No se encontró la habitación" });
      }
      } catch (error) {
        res.status(400).json({ error: error.message });
      }
})
```

- **PUT /api/habitacion/:numero**
  Actualiza la información de una habitación específica (identificada por **numeroHabitacion**).

```
//actualiza la habitación por el id del parámetro
router.put('/habitacion/:numero', async (req, res) => {
    try {
      const habitacion = await Habitacion.updateOne(
        {numeroHabitacion: req.params.numero},
        {$set: req.body}
      )
      console.log(habitacion)
      if (habitacion.modifiedCount > 0) {
        res.json({ message: "Habitación actualizada correctamente" });
    } else {
        res.status(404).json({ message: "No se encontró la habitación o no hubo cambios" });
    }
    } catch (error) {
      res.status(400).json({ error: error.message });
    }
  });
```

- **GET /api/habitacionPisoMax**
  Retorna el pisomás alto que existe entre todas las habitaciones (usando aggregate de MongoDB).

```
router.get('/habitacionPisoMax', async (req, res) => {
  try{const pisoMax = await Habitacion.aggregate([
    { $sort: { "piso": -1 } },
    { $limit: 1 },
  { $project: { _id: 0, "piso": 1 } }
  ])
  res.status(200).json(pisoMax[0].piso)
}
  catch (error){
    res.status(400).json({ error: error.message })
  }
})
```

- **GET /api/habitacionPrecioMax**
  Retorna el precio más alto que existe entre todas las habitaciones.

- ```
router.get('/habitacionPrecioMax', async (req, res) => {
  try{const PrecioMax = await Habitacion.aggregate([
    { $sort: { "precio": -1 } },
    { $limit: 1 },
  { $project: { _id: 0, "precio": 1 } }
  ])
  res.status(200).json(PrecioMax[0].precio)
}
  catch (error){
    res.status(400).json({ error: error.message })
  }
})
```

Adicionalmente:

- **GET /api/tipoHabitaciones**
  Obtiene todos los tipos de habitación disponibles (modelo **TipoHabitacion**).
  - ```
router.get('/tipoHabitaciones', async (req, res) => {
  try{const TipoHabitaciones = await TipoHabitacion.find()
  res.status(200).json(TipoHabitaciones)}
  catch (error){
    res.status(400).json({ error: error.message })
  }
})
```

Modelos de Datos (Mongoose)

- **TipoHabitacion** (almacena la estructura base de un tipo de habitación, su precio, capacidad de camas y capacidad de personas).
- **Habitacion** (almacena información detallada de cada habitación, como número, descripción, precio, disponibilidad, piso, rutas de fotos y el subdocumento **tipoHabitacion**).

Conexión a la Base de Datos

- Se realiza con **mongoose.connect(mongoString)**, donde **mongoString** proviene de la variable de entorno **DATABASE_URL**.
- Al conectarse, se muestra en consola si la conexión es exitosa o si ocurre algún error.
  - ```
mongoose.connect(mongoString)
.then(console.log('Conexión BDD existosa'))
.catch(error => console.log('Error al intentar conectar con la BDD', error))
```
```

Uso de Variables de Entorno

- El archivo **.env** maneja valores críticos como **PORT** (puerto donde corre el servidor) y **DATABASE_URL** (URL de conexión a MongoDB).