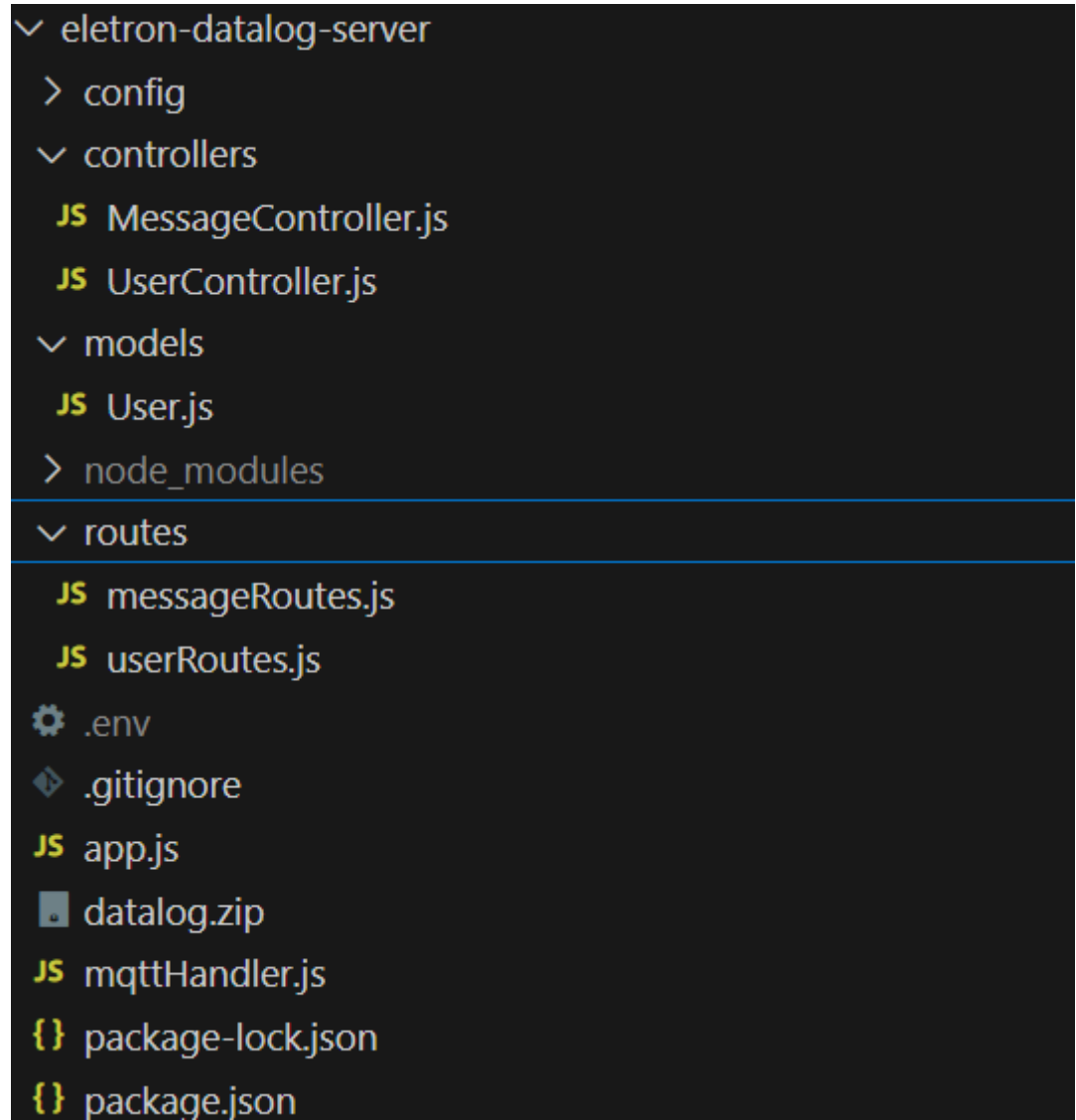


Temos a seguinte aplicação em backend node.js

eletron-datalog-server



```

▼ eletron-datalog-server
  > config
  ▼ controllers
    JS MessageController.js
    JS UserController.js
  ▼ models
    JS User.js
  > node_modules
  ▼ routes
    JS messageRoutes.js
    JS userRoutes.js
  ⚙ .env
  ⚙ .gitignore
  JS app.js
  📁 datalog.zip
  JS mqttHandler.js
  {} package-lock.json
  {} package.json

```

The image shows a file explorer view of the 'eletron-datalog-server' project. The directory structure is as follows:

- eletron-datalog-server
 - config
 - controllers
 - MessageController.js
 - UserController.js
 - models
 - User.js
 - node_modules
 - routes
 - messageRoutes.js
 - userRoutes.js
 - .env
 - .gitignore
 - app.js
 - datalog.zip
 - mqttHandler.js
 - package-lock.json
 - package.json

Config

db.js

```
import mongoose from 'mongoose';
```

```
import dotenv from 'dotenv';
```

```
dotenv.config(); // Carrega as variáveis de ambiente do .env
```

```
const connectDB = async () => {
```

```
  try {
```

```
    mongoose.connect(`mongodb+srv://${process.env.DB_USER}:${process.env.DB_PASSWORD}@eletron-cluster.frijdfp.mongodb.net/datalog?retryWrites=true&w=majority&appName=eletron-cluster` ).then().catch((err) => {console.log(err)});
```

```
    console.log('MongoDB connected!');
```

```
  } catch (error) {
```

```
    console.error(` Error connecting to MongoDB: ${error.message} `);
```

```
    process.exit(1); // Encerra a aplicação em caso de erro no banco
```

```
  }
```

```
};
```

```
export default connectDB;
```

Controllers

MessageController.js

```
const MessageController = {
```

```
getMessages(req, res) {  
  const messages = req.app.locals.messages || [];  
  res.json(messages);  
},  
};
```

```
export default MessageController;
```

UserController.js

```
import bcrypt from 'bcrypt';  
import jwt from 'jsonwebtoken';  
import User from '../models/User.js';  
  
const UserController = {  
  async register(req, res) {  
    const { name, email, password, confirmpassaword } = req.body;  
  
    if (!email || !password || !confirmpassaword) {  
      return res.status(400).json({ message: 'Todos os campos são obrigatórios!' });  
    }  
  
    if (password !== confirmpassaword) {  
      return res.status(400).json({ message: 'As senhas não conferem!' });  
    }  
  
    try {
```

```
const userExists = await User.findOne({ email });
```

```
if (userExists) {
```

```
  return res.status(400).json({ message: 'E-mail já cadastrado!' });
```

```
}
```

```
const hashedPassword = await bcrypt.hash(password, 10); // Gera hash da  
senha
```

```
const user = new User({ name, email, password: hashedPassword });
```

```
await user.save();
```

```
res.status(201).json({ message: 'Usuário cadastrado com sucesso!' });
```

```
} catch (error) {
```

```
  res.status(500).json({ message: 'Erro ao salvar usuário.' });
```

```
}
```

```
},
```

```
async login(req, res) {
```

```
  const { email, password } = req.body;
```

```
  if (!email || !password) {
```

```
    return res.status(400).json({ message: 'Todos os campos são obrigatórios!' });
```

```
}
```

```
try {
```

```
  const user = await User.findOne({ email });
```

```
  if (!user || !(await bcrypt.compare(password, user.password))) {
```

```
    return res.status(400).json({ message: 'Credenciais inválidas!' });
```

```
}
```

```
const token = jwt.sign({ id: user._id }, process.env.SECRET, { expiresIn: '30d' });
```

```
res.status(200).json({ message: 'Autenticação bem-sucedida!', token, user:  
{ name: user.name } });
```

```
} catch (error) {
```

```
res.status(500).json({ message: 'Erro ao autenticar usuário.' });
```

```
}
```

```
},
```

```
async getUser(req, res) {
```

```
try {
```

```
const user = await User.findById(req.params.id).select('-password');
```

```
if (!user) {
```

```
return res.status(404).json({ message: 'Usuário não encontrado!' });
```

```
}
```

```
res.json(user);
```

```
} catch (error) {
```

```
res.status(500).json({ message: 'Erro ao buscar usuário.' });
```

```
}
```

```
},
```

```
};
```

```
export default UserController;
```

models

User.js

```
import mongoose from 'mongoose';
```

```
const UserSchema = new mongoose.Schema({  
  name: { type: String, required: true, trim: true },  
  email: { type: String, required: true, unique: true, trim: true },  
  password: { type: String, required: true },  
  lastLogin: { type: Date, default: null },  
});
```

```
const User = mongoose.model('User', UserSchema);  
export default User;
```

routes

messageRouters.js

```
import express from 'express';  
import messageController from '../controllers/MessageController.js';
```

```
const router = express.Router();
```

```
router.get('/', messageController.getMessages);
```

```
export default router;
```

userRoutes.js

```
import express from 'express';
import userController from '../controllers/UserController.js';

const router = express.Router();

// Rotas relacionadas ao usuário
router.post('/register', userController.register);
router.post('/login', userController.login);
router.get('/:id', userController.getUser);

export default router;
```

app.js

```
import express from 'express';
import dotenv from 'dotenv';
import cors from 'cors';
import connectDB from './config/db.js';
import userRoutes from './routes/userRoutes.js';
import messageRoutes from './routes/messageRoutes.js';
import mqttHandler from './mqttHandler.js';

dotenv.config();
```

```
const app = express();
```

```
const allowedOrigins = [  
  'http://localhost:5173',  
  'https://main.d1o387bagj6v4q.amplifyapp.com'  
];
```

```
const corsOptions = {  
  origin: (origin, callback) => {  
    // Permitir sem origin (ex: Postman, server2server)  
    if (!origin) return callback(null, true);  
    // Permitir localhost e amplify  
    if (allowedOrigins.includes(origin)) return callback(null, true);  
    // Bloquear outras origens  
    return callback(new Error('Not allowed by CORS'));  
  },  
  credentials: true,  
  methods: ['GET','POST','PUT','DELETE','OPTIONS'],  
  allowedHeaders: ['Content-Type', 'Authorization'],  
  optionsSuccessStatus: 200  
};
```

```
app.use(cors(corsOptions));
```

```
// Middleware para garantir que a rota OPTIONS funcione (preflight)
```

```
app.use(express.json()); // Middleware para JSON
```



```
// Conecta ao banco de dados
```

```
connectDB();
```

```
// Configura MQTT (broker)
```

```
mqttHandler(app); // Configuração de mensagens em tempo real
```

```
// Configuração das rotas
```

```
app.use('/auth', userRoutes); // Rota de autenticação de usuário
```

```
app.use('/api/mensagens', messageRoutes); // Rota de dados MQTT
```

```
// Endpoint inicial para verificar status
```

```
app.get('/', (req, res) => {
```

```
  res.status(200).json({ message: 'Hello World!' });
```

```
});
```

```
// Inicia o servidor
```

```
const port = process.env.PORT || 3000;
```

```
app.listen(port, () => {
```

```
  console.log(` Servidor rodando na porta ${port} `);
```

```
});
```

```
mqttHandler.js
```

```
import mqtt from 'mqtt';
```

```
const messages = []; // Armazena as mensagens temporariamente
```

```
const mqttHandler = (app) => {

  const mqttClient =
  mqtt.connect('mqtt://29232f271b9f47cb8d55000d4557bc0c.s1.eu.hivemq.cloud
:8883', {

    username: 'ESP32',

    password: 'Eletron0101',

  });

  mqttClient.on('connect', () => {

    console.log('Connected to MQTT broker');

    mqttClient.subscribe('maquina/dados');

  });

  mqttClient.on('message', (topic, message) => {

    try {

      const data = JSON.parse(message.toString());

      console.log(` Received message on ${topic}:`, data);

      messages.push(data); // Armazena a mensagem

    } catch (error) {

      console.error('Failed to parse MQTT message:', error);

    }

  });

  // Disponibiliza as mensagens no servidor Express

  app.locals.messages = messages;

};

export default mqttHandler;
```

