Backoffice/Server/server.js

```
const express = require('express');
const cors = require('cors');
const admin = require('firebase-admin');
const bcrypt = require('bcrypt');
const serviceAccount = require('./ual-tech-firebase-adminsdk-p5fbv-f1103dd8a0.json');

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});

const db = admin.firestore();
db.settings({
  ignoreUndefinedProperties: true
});
const FieldValue = admin.firestore.FieldValue;
const app = express();
const SERVER = 'emac75.ddns.net'
const PORT = 5556;

app.use(cors());
app.use(express.json());


app.post('/login', async (req, res) => {
  const { email, password } = req.body;

  try {
      const usersRef = db.collection('Users');
      const snapshot = await usersRef.where('email', '==', email).get();

      if (snapshot.empty) {
          return res.status(401).send({ message: 'E-mail ou palavra-passe inválidos 1' });
      }

      let user;
      snapshot.forEach(doc => {
          user = { id: doc.id, ...doc.data() };
      });

      const isPasswordValid = await bcrypt.compare(password, user.password);
      if (!isPasswordValid) {
          return res.status(401).send({ message: 'E-mail ou palavra-passe inválidos 2' });
      }

      if (user.admin === true) {
          return res.status(200).send({ message: 'Login com sucesso', user });
      } else {
          return res.status(403).send({ message: 'O utilizador não tem privilégios de administrador' });
```

```
      }
  } catch (error) {
      console.error('Erro no registo', error);
      return res.status(500).send('Erro interno do servidor');
  }
});

app.get('/users', async (req, res) => {
  try {
    const usersRef = db.collection('Users');
    const snapshot = await usersRef.get();
    const users = [];
    snapshot.forEach(doc => {
      users.push({ id: doc.id, ...doc.data() });
    });
    res.json(users);
  } catch (error) {
    console.error('Erro ao obter documentos', error);
    res.status(500).send('Erro ao recuperar utilizadores');
  }
});

app.put('/users/:userId', async (req, res) => {
  const { userId } = req.params;
  const { username, email, password, admin } = req.body;
  console.log(username, email, password, admin);
  try {
    if (password) {
      console.log("COM")
      const hashedPassword = await bcrypt.hash(password, 12);
      await db.collection('Users').doc(userId).update({
        username,
        email,
        password: hashedPassword,
        admin
      });
    } else {
      console.log("SEM")
      await db.collection('Users').doc(userId).update({
        username,
        email,
        admin
      });
    }
    res.status(200).send('Utilizador atualizado com sucesso');
  } catch (error) {
    console.error('Erro ao atualizar o utilizador', error);
    res.status(500).send('Erro ao atualizar o utilizador');
  }
});

app.put('/users/:userId/delete-fields', async (req, res) => {
  const { userId } = req.params;

```

```
103   try {
104     await db.collection('Users').doc(userId).update({
105       cameradata: FieldValue.delete(),
106       voicefeatures: FieldValue.delete()
107     });
108     res.status(200).send('Campos eliminados com êxito');
109   } catch (error) {
110     console.error('Erro ao eliminar campos', error);
111     res.status(500).send('Erro ao eliminar campos');
112   }
113 });
114 
115 app.delete('/users/:userId', async (req, res) => {
116   const { userId } = req.params;
117   try {
118     await db.collection('Users').doc(userId).delete();
119     res.status(200).send('Utilizador eliminado com sucesso');
120   } catch (error) {
121     console.error('Erro ao eliminar o utilizador', error);
122     res.status(500).send('Erro ao eliminar o utilizador');
123   }
124 });
125 
126 app.listen(PORT, () => {
127   console.log(`Servidor em execução em http://${SERVER}:${PORT}`);
128 });
```