Code

git add .

git commit -m "Votre message de mise à jour"

git push origin main

taskkill /F /IM java.exe ( le processus "java.exe" de PID 13544 a été arrêté.)

<access origin="\*" />

    <allow-navigation href="https://farmsconnect-b084ddb02391.herokuapp.com" />

    <allow-intent href="geo:\*" />

    <preference name="GeolocationEnabled" value="true" />

    <platform name="android">

    <content src="https://farmsconnect-b084ddb02391.herokuapp.com" />

    <access origin="https://farmsconnect-b084ddb02391.herokuapp.com" />

    <allow-navigation href="https://farmsconnect-b084ddb02391.herokuapp.com" />

    <allow-intent href="geo:\*" />

    <allow-intent href="http://\*/\*" />

    <allow-intent href="https://\*/\*" />

<preference name="AndroidWindowSplashScreenAnimatedIcon" value="resources/splash.png" />

<preference name="AndroidWindowSplashScreenDuration" value="5000" />

<preference name="AndroidWindowSplashScreenBackgroundColor" value="#d0e2f7" />

<preference name="SplashScreen" value="screen" />

<preference name="SplashScreenDelay" value="3000" />

<preference name="SplashScreenFadeOutDuration" value="1500" />

    <platform name="android">

        <icon density="ldpi" src="resources/android/icon/drawable-ldpi-icon.png" />

        <icon density="mdpi" src="resources/android/icon/drawable-mdpi-icon.png" />

        <icon density="hdpi" src="resources/android/icon/drawable-hdpi-icon.png" />

        <icon density="xhdpi" src="resources/android/icon/drawable-xhdpi-icon.png" />

        <icon density="xxhdpi" src="resources/android/icon/drawable-xxhdpi-icon.png" />

        <icon density="xxxhdpi" src="resources/android/icon/drawable-xxxhdpi-icon.png" />

    <feature name="Geolocation">

        <param name="android-package" value="android.location.Location"/>

    </feature>

    <preference name="GeolocationEnabled" value="true" />

    <config-file parent="/manifest" target="AndroidManifest.xml">

        <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

        <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

    </config-file>

    </platform>

</widget>

heroku logs –tail

server.js fontionnel   
  
  
«  // Importer les dépendances

const express = require('express');

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const cors = require('cors');

const bodyParser = require('body-parser');

const nodemailer = require('nodemailer');

const http = require('http');

const socketIo = require('socket.io');

const multer = require('multer');

const path = require('path');

// Modèles

const User = require('./models/User');

const Message = require('./models/message');

const Annonce = require('./models/Annonce'); // Assurez-vous que le chemin du modèle est correct

// Créer une instance de l'application Express

const app = express();

const server = http.createServer(app);

const io = socketIo(server);

// Middleware pour augmenter la limite de taille des requêtes JSON

app.use(express.json({ limit: '20mb' })); // Vous pouvez ajuster cette valeur

// Middleware pour analyser le corps des requêtes JSON

app.use(express.json());

app.use(bodyParser.json());

// Middleware pour servir des fichiers statiques

app.use(express.static('public'));

// Middleware CORS

app.use(cors({

    origin: 'https://farmsconnect-b084ddb02391.herokuapp.com', // ou '\*' pour autoriser toutes les origines

    methods: ['GET', 'POST', 'PUT', 'DELETE'],

}));

// Connexion à MongoDB

mongoose.connect('mongodb+srv://kabboss:ka23bo23re23@cluster0.uy2xz.mongodb.net/FarmsConnect?retryWrites=true&w=majority', {})

    .then(() => console.log('Connecté à MongoDB...'))

    .catch(err => console.error('Erreur de connexion à MongoDB:', err));

// Configurer le transporteur Nodemailer

const transporter = nodemailer.createTransport({

    service: 'gmail',

    auth: {

        user: 'kaboreabwa2020@gmail.com',

        pass: 'swbo vejr klic otpu' // Remplacez par votre mot de passe ou mot de passe d'application

    }

});

// Routes

// Route pour s'inscrire (signup)

app.post('/api/signup', async (req, res) => {

    const { username, email, contact, password, userType } = req.body;

    try {

        const existingUser = await User.findOne({ username });

        if (existingUser) {

            return res.status(400).send('Cet utilisateur existe déjà.');

        }

        const hashedPassword = await bcrypt.hash(password, 10);

        const newUser = new User({ username, email, contact, password: hashedPassword, userType });

        await newUser.save();

        res.status(201).send('Utilisateur créé avec succès !');

    } catch (error) {

        console.error('Erreur lors de l’inscription :', error);

        res.status(500).send('Erreur lors de l’inscription : ' + error.message);

    }

});

// Route pour se connecter (login)

app.post('/api/login', async (req, res) => {

    const { username, email, contact, password } = req.body;

    try {

        if (!username || !email || !contact || !password) {

            return res.status(400).send('Veuillez fournir le nom d’utilisateur, l\'email, le contact et le mot de passe.');

        }

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

        if (!user) {

            return res.status(400).send('Nom d’utilisateur incorrect.');

        }

        if (user.email !== email || user.contact !== contact) {

            return res.status(400).send('Email ou contact incorrect.');

        }

        const isMatch = await bcrypt.compare(password, user.password);

        if (!isMatch) {

            return res.status(400).send('Mot de passe incorrect.');

        }

        const token = jwt.sign({ userId: user.\_id }, 'ka23bo23re23'); // Remplacez par une vraie clé secrète

        res.status(200).json({ username: user.username, email: user.email, contact: user.contact, token, message: 'Connexion réussie !' });

    } catch (error) {

        console.error('Erreur lors de la connexion :', error);

        res.status(500).send('Erreur lors de la connexion : ' + error.message);

    }

});

// Route pour passer une commande et envoyer l'email

app.post('/api/order', async (req, res) => {

    console.log('Données reçues :', req.body);

    const { username, email, contact, price, quantity, weight, Produit: nomproduit } = req.body;

    if (!username || !nomproduit || !email || !contact || !price || !quantity || !weight) {

        console.error('Informations manquantes dans la requête :', req.body);

        return res.status(400).send('Veuillez fournir toutes les informations nécessaires : username, nom du produit, email, contact, prix, quantité et poids.');

    }

    try {

        const mailOptions = {

            from: 'kaboreabwa2020@gmail.com',

            to: email,

            subject: 'Confirmation de commande',

            text: `Merci, ${username}, pour votre commande !\n\nVoici les détails :\n- Produit : ${nomproduit}\n- Prix total : ${price} FCFA\n- Quantité : ${quantity}\n- Poids total : ${weight} kg\n\nNous vous contacterons au numéro ${contact} pour valider la commande.\n\nNB : Les animaux subiront un contrôle de santé avec l'un de nos vétérinaires, et vous serez livré dans un délai de 24 heures maximum. Le paiement se fera à la livraison.`

        };

        transporter.sendMail(mailOptions, (error, info) => {

            if (error) {

                console.error('Erreur lors de l\'envoi de l\'e-mail :', error);

                return res.status(500).send('Erreur lors de l\'envoi de l\'e-mail : ' + error.message);

            }

            console.log('E-mail envoyé : ' + info.response);

            res.status(200).send('Commande passée avec succès et e-mail envoyé !');

        });

    } catch (error) {

        console.error('Erreur lors du traitement de la commande :', error);

        res.status(500).send('Erreur lors du traitement de la commande : ' + error.message);

    }

});

// Route pour servir le fichier users.html

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

    res.sendFile(\_\_dirname + '/public/users.html');

});

// Route pour servir le fichier Visiteur.html

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

    res.sendFile(\_\_dirname + '/public/Visiteur.html');

});

// Route pour récupérer les informations de l'utilisateur

app.post('/api/getUserInfo', async (req, res) => {

    const userId = req.body.userId;

    try {

        const user = await User.findById(userId);

        res.json(user);

    } catch (error) {

        res.status(500).json({ error: 'Erreur lors de la récupération des informations de l\'utilisateur.' });

    }

});

// Route pour envoyer les informations d'achat par email

app.post('/api/send-email', async (req, res) => {

    const { content } = req.body;

    const mailOptions = {

        from: 'kaboreabwa2020@gmail.com',

        to: 'kaboreabwa2020@gmail.com',

        subject: 'Nouvelle commande reçue',

        text: content

    };

    transporter.sendMail(mailOptions, (error, info) => {

        if (error) {

            console.error('Erreur lors de l\'envoi de l\'e-mail :', error);

            return res.status(500).send('Erreur lors de l\'envoi de l\'e-mail.');

        }

        console.log('E-mail envoyé :', info.response);

        res.status(200).send('E-mail envoyé avec succès.');

    });

});

// Route pour ajouter une annonce

app.post('/api/annonces', async (req, res) => {

    try {

        const annonce = new Annonce(req.body);

        await annonce.save(); // Sauvegarde l'annonce dans MongoDB

        res.status(201).json({ message: 'Annonce ajoutée avec succès' });

    } catch (error) {

        console.error(error);

        res.status(500).json({ message: 'Erreur lors de l\'ajout de l\'annonce' });

    }

});

// Route pour récupérer toutes les annonces

app.get('/api/annonces', async (req, res) => {

    try {

        const annonces = await Annonce.find(); // Récupère toutes les annonces depuis MongoDB

        res.json(annonces);

    } catch (error) {

        console.error(error);

        res.status(500).json({ message: 'Erreur lors du chargement des annonces' });

    }

});

// Routes pour les messages

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

    Message.find()

        .then(messages => res.json(messages))

        .catch(err => res.status(400).json(err));

});

app.post('/api/messages', (req, res) => {

    const { username, content } = req.body;

    const newMessage = new Message({ username, content });

    newMessage.save()

        .then(message => {

            res.status(201).json(message);

            // Émettre l'événement pour les nouveaux messages

            io.emit('newMessage', message);

        })

        .catch(err => res.status(500).json({ error: err.message }));

});

app.put('/api/messages/:id', (req, res) => {

    const { content } = req.body;

    Message.findByIdAndUpdate(req.params.id, { content }, { new: true })

        .then(updatedMessage => res.json(updatedMessage))

        .catch(err => res.status(400).json(err));

});

app.delete('/api/messages/:id', (req, res) => {

    Message.findByIdAndDelete(req.params.id)

        .then(() => res.status(204).send())

        .catch(err => res.status(400).json(err));

});

// Routes pour les réponses

app.post('/api/messages/:id/replies', (req, res) => {

    const { username, content } = req.body;

    const reply = { username, content };

    Message.findByIdAndUpdate(req.params.id, { $push: { replies: reply } }, { new: true })

        .then(updatedMessage => {

            io.emit('newMessage', updatedMessage);

            res.json(updatedMessage);

        })

        .catch(err => res.status(400).json(err));

});

// Routes pour les utilisateurs

app.post('/api/users', (req, res) => {

    const { username, email, contact } = req.body;

    const newUser = new User({ username, email, contact });

    newUser.save()

        .then(savedUser => res.json(savedUser))

        .catch(err => res.status(400).json(err));

});

//Formation

// Configuration du serveur

const PORT = process.env.PORT || 3002;

server.listen(PORT, () => {

    console.log(`Serveur en écoute sur le port ${PORT}`);

});

// WebSocket pour les communications en temps réel

io.on('connection', (socket) => {

    console.log('Nouvel utilisateur connecté');

    socket.on('disconnect', () => {

        console.log('Utilisateur déconnecté');

    });

});