# Authentification Token

Authentication
Authorization
USER NAME
login
Confirms users are who they say they are.
Gives users permission to access a resource.

## Ajouter le models user

```
const mongoose = require('mongoose')


const userSchema = new mongoose.Schema({

    email: {
        type: String,
        required: true,
        unique: true
    },

    password: {
        type: String,
        required: true,
    },

    firstname: {
        type: String,
        required: true,
    },
    lastname: {
        type: String,
        required: true,
    },

    role: {
        type: String,
        enum: ["user", "admin"],
        default: "user"
    },
isActive: {
        type: Boolean,
        default: true,
        required: false
    },

   avatar :{

        type: String,

        required: false

        }    ,


},
{
    timestamps: true,
},
)

module.exports = mongoose.model('User', userSchema)
```

## Créer le fichier user.route.js

Ajouter la méthode de création d'un utilisateur

```
const express = require('express');
const router = express.Router();
const User=require("../models/user")
```

```
// créer un nouvel utilisateur

router.post('/register', async (req, res) =>  {
 try {

        let { email, password, firstname, lastname } = req.body
        const user = await User.findOne({ email })
        if (user) return res.status(404).send({ success: false, message:
"User already exists" })


        const newUser = new User({ email, password, firstname, lastname })

        const createdUser = await newUser.save()

        return res.status(201).send({ success: true, message: "Account
created successfully", user: createdUser })

    } catch (err) {
        console.log(err)
        res.status(404).send({ success: false, message: err })

    }

});
module.exports = router;
```

Ajouter la méthode de cryptage de mot de passe avant enregistrement (save)

```
const mongoose = require('mongoose')

const bcrypt = require('bcrypt')

const userSchema = new mongoose.Schema({

    email: {
        type: String,
        required: true,
        unique: true
    },

    password: {
        type: String,
        required: true,
    },

    firstname: {
        type: String,
        required: true,
    },

    lastname: {
        type: String,
        required: true,
    },

    role: {
        type: String,
        enum: ["user", "admin"],
        default: "user"
    },


},
{
    timestamps: true,
},
)
userSchema.pre('save', async function(next) {

    if (!this.isModified('password')) return next()
    const salt = await bcrypt.genSalt(10)
    const hashedPassword = await bcrypt.hash(this.password, salt)
    this.password = hashedPassword
    next()

})
module.exports = mongoose.model('User', userSchema)
```

créer la méthode qui affiche la liste des utilisateurs

```
// afficher la liste des utilisateurs.

router.get('/', async (req, res, )=> {

    try {

        const users = await User.find().select("-password");

        res.status(200).json(users);

    } catch (error) {

        res.status(404).json({ message: error.message });

    }


});
```

## Ajouter la méthode qui modifie l'état du champ iSactive

```
/**
 * as an admin i can disable or enable an account
 */
   router.get('/status/edit/',  async (req, res) =>  {
      try {

          let  email  = req.query.email
          let user = await User.findOne({email})
          user.isActive = !user.isActive
          user.save()
          res.status(200).send({ success: true, user })
      } catch (err) {
          return res.status(404).send({ success: false, message: err })
      }
     })
```

## Installer jsonwebtoken et bcrypt

```
npm i jsonwebtoken bcrypt
```

ajouter les imports suivants dans le fichier routes/user.route.js

```
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
```

## Ajouter le code de connexion suivant

```
// se connecter
router.post('/login', async (req, res) =>  {
    try {
        let { email, password } = req.body


        if (!email || !password) {
            return res.status(404).send({ success: false, message: "All
fields are required" })
```

```
        }

        let user = await User.findOne({ email
}).select('+password').select('+isActive')


        if (!user) {

            return res.status(404).send({ success: false, message: "Account
doesn't exists" })

        } else {

      let isCorrectPassword = await bcrypt.compare(password, user.password)
       if (isCorrectPassword) {

                delete user._doc.password
                if (!user.isActive) return res.status(200).send({ success:
false, message: 'Your account is inactive, Please contact your
administrator' })

                const token = jwt.sign ({ iduser:
user._id,name:user.firstname, role: user.role }, process.env.SECRET, {
expiresIn: "1h", })

                return res.status(200).send({ success: true, user, token })

            } else {

                return res.status(404).send({ success: false, message:
"Please verify your credentials" })

            }
        }

    } catch (err) {
        return res.status(404).send({ success: false, message: err.message
})
    }

   });
```

## Ajouter dans le fichier .env le mot secret

```
ENV=DEVELOPMENT

DATABASE=mongodb://127.0.0.1:27017/DatabaseCommerce

DATABASECLOUD=mongodb+srv://user:password@cluster0.yfhhu.mongodb.net/Databas

eCommerce?retryWrites=true&w=majority

PORT=3001

SECRET = 3283931617424356 34
```

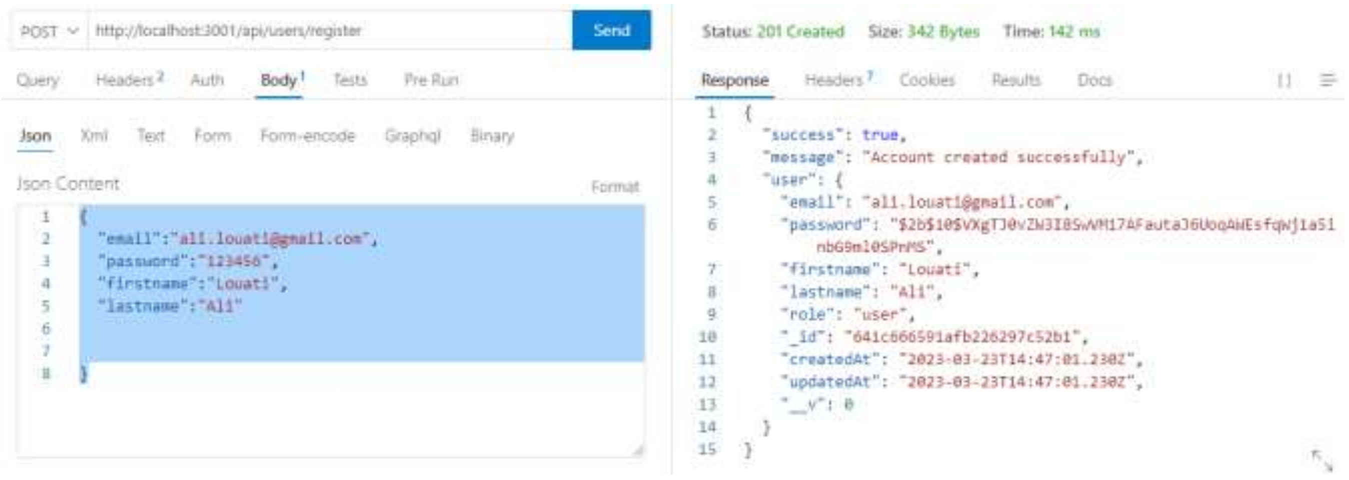
Ajouter les routes de user dans le fichier app.js

```
app.use('/api/users', userRouter);
const userRouter =require("./routes/user.route")
```

Test : register :

Post : http://localhost:3001/api/users/register

```
{
  "email":"ali.louati@gmail.com",
  "password":"123456",
  "firstname":"Louati",
  "lastname":"Ali"
}
```



```
http://localhost:3001/api/users/register
```

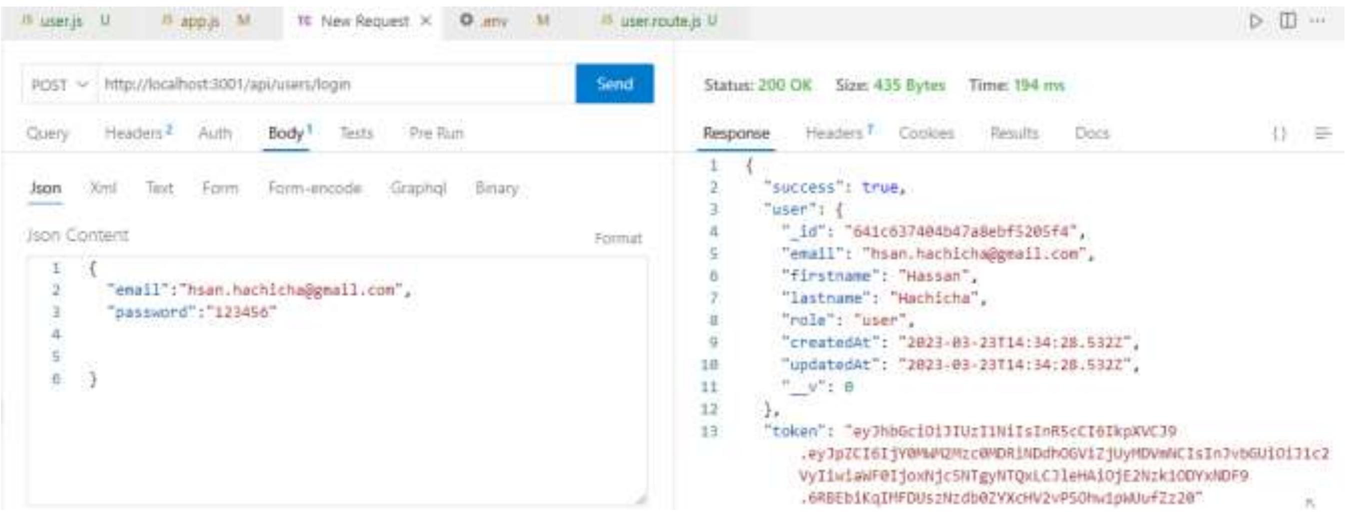
```
{
  "email":"ali.louati@gmail.com",
  "password":"123456",
  "firstname":"Louati",
  "lastname":"Ali"


}
```

```
Status: 201 Created    Size: 342 Bytes    Time: 142 ms
```

```
{
  "success": true,
  "message": "Account created successfully",
  "user": {
    "email": "ali.louati@gmail.com",
    "password": "$2b$10$VXgTJ0vZW3I8SwM17AFautaJ6UoqAWEsfqWj1a51
        nbG9ml0SPnMS",
    "firstname": "Louati",
    "lastname": "Ali",
    "role": "user",
    "_id": "641c666591afb226297c52b1",
    "createdAt": "2023-03-23T14:47:01.230Z",
    "updatedAt": "2023-03-23T14:47:01.230Z",
    "__v": 0
  }
}
```

Test : login

Post : http://localhost:3001/api/users/login



```
http://localhost:3001/api/users/login
```

```
{
  "email":"hsan.hachicha@gmail.com",
  "password":"123456"


}
```

```
Status: 200 OK    Size: 435 Bytes    Time: 194 ms
```

```
{
  "success": true,
  "user": {
    "_id": "641c637404b47a8ebf5205f4",
    "email": "hsan.hachicha@gmail.com",
    "firstname": "Hassan",
    "lastname": "Hachicha",
    "role": "user",
    "createdAt": "2023-03-23T14:34:28.532Z",
    "updatedAt": "2023-03-23T14:34:28.532Z",
    "__v": 0
  },
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJpZCI6IjY0MWM2Mzc0MDRiNDdhOGViZjUyMDVmNCIsInJvbGUiOiJ1c2
      VyIiwiaWF0IjoxNjc5NTgyNTQxLCJleHAiOjE2Nzk1ODYxNDF9
      .6RBEbiKqIMFDUszNzdb0ZYXcHV2vP5Ohw1pWUufZz20"
```

```
git add .
git commit -m "feat:login user:login"
git push
git status
```

Middleware

Ajouter les middleware

verify-token : jsonwebtoken

verify-role : pour verifier le role user ou Admin

upload-file(multer)

1- Verif-token
   Créer le dossier middleware
   Céer le fichier verify-token.js

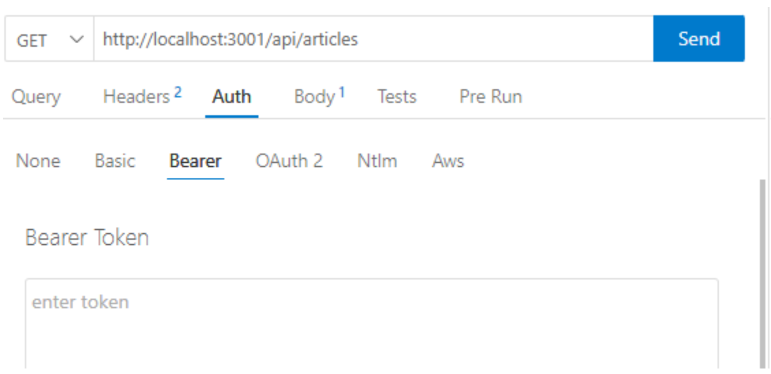
```
const jwt = require('jsonwebtoken');
require('dotenv').config()

/**
 *
 * Middleware function that check if user has token and can access or not
 *
 * @param {string} token - Authorization token provided by request sender
 *
 * @return
 * @status 403 No token provided
 * @Status 403 Invalid token
 *
 */

const verifyToken = (req, res, next) => {

    const header = req.headers['authorization'];
    const token = header && header.split('Bearer ')[1];
    if (!token) return res.status(403).send({ success: false, message: 'No
token provided' });

    jwt.verify(token, process.env.SECRET, (err, decoded) => {

        if (err) return res.status(403).send({ success: false, message:
'Invalid token' });
        req.user = {}
        req.user.id = decoded.iduser
        req.user.role = decoded.role
        next()
    })
}



module.exports = { verifyToken }
```

Dans le fichier article.route.js utiliser la fonction callback veriftoken()

```
const {verifyToken} =require("../middleware/veriftoken")
// afficher la liste des articles.
router.get('/', verifyToken, async (req, res )=> {
    try {
        const articles = await Article.find();
        res.status(200).json(articles);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
});
```

Test

Afficher la liste des articles sans avopir le token (No token provided)



http://localhost:3001/api/articles

Status: 403 Forbidden Size: 47 Bytes Time: 5 ms

Bearer Token

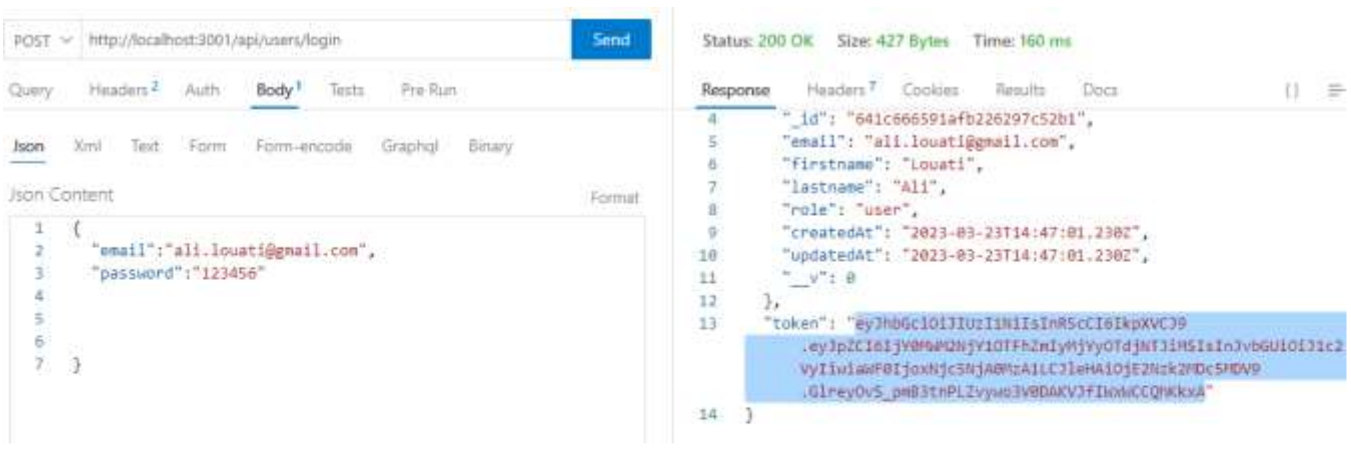
enter token

```
{
  "success": false,
  "message": "No token provided"
}
```

Il faut donc faire un login avant (se connecter)

login pour se connecter et avoir le token

post : http://localhost:3001/api/users/login



http://localhost:3001/api/users/login

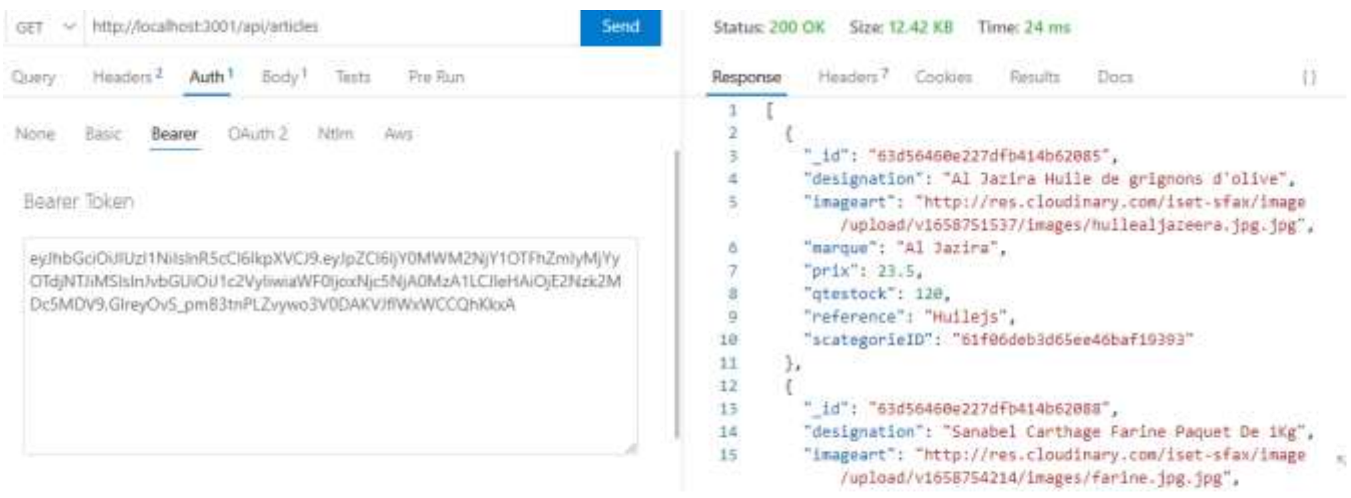
Json Content

```
{
  "email":"ali.louati@gmail.com",
  "password":"123456"




}
```

Status: 200 OK Size: 427 Bytes Time: 160 ms

```
    "_id": "641c666591afb226297c52b1",
    "email": "ali.louati@gmail.com",
    "firstname": "Louati",
    "lastname": "Ali",
    "role": "user",
    "createdAt": "2023-03-23T14:47:01.230Z",
    "updatedAt": "2023-03-23T14:47:01.230Z",
    "__v": 0
  },
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJpZCI6IjY0MWM2NjY1OTFhZmIyMjYyOTdjNTJiMSIsInJvbGUiOiJ1c2
      VyIiwiaWF0IjoxNjc5NjA0MzA1LCJleHAiOjE2Nzk2MDc5MDV9
      .GlreyOvS_pmB3tnPLZvywo3V0DAKVJfIWxWCCQhKkxA"
}
```

Copier le token dans la clause Headers



http://localhost:3001/api/articles

Bearer Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY0MWM2NjY1OTFhZmIyMjYyOTdjNTJiMSIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNjc5NjA0MzA1LCJleHAiOjE2Nzk2MDc5MDV9.GlreyOvS_pmB3tnPLZvywo3V0DAKVJfIWxWCCQhKkxA

Status: 200 OK Size: 12.42 KB Time: 24 ms

```
[
  {
    "_id": "63d56460e227dfb414b62085",
    "designation": "Al Jazira Huile de grignons d'olive",
    "imageart": "http://res.cloudinary.com/iset-sfax/image
       /upload/v1658751537/images/huilealjazeera.jpg.jpg",
    "marque": "Al Jazira",
    "prix": 23.5,
    "qtestock": 120,
    "reference": "Huilejs",
    "scategorieID": "61f06deb3d65ee46baf19393"
  },
  {
    "_id": "63d56460e227dfb414b62088",
    "designation": "Sanabel Carthage Farine Paquet De 1Kg",
    "imageart": "http://res.cloudinary.com/iset-sfax/image
       /upload/v1658754214/images/farine.jpg.jpg",
```

2- Multer

créer dans le dossier middleware le fichier uploadfile.js

Installer multer

```
npm install multer
```

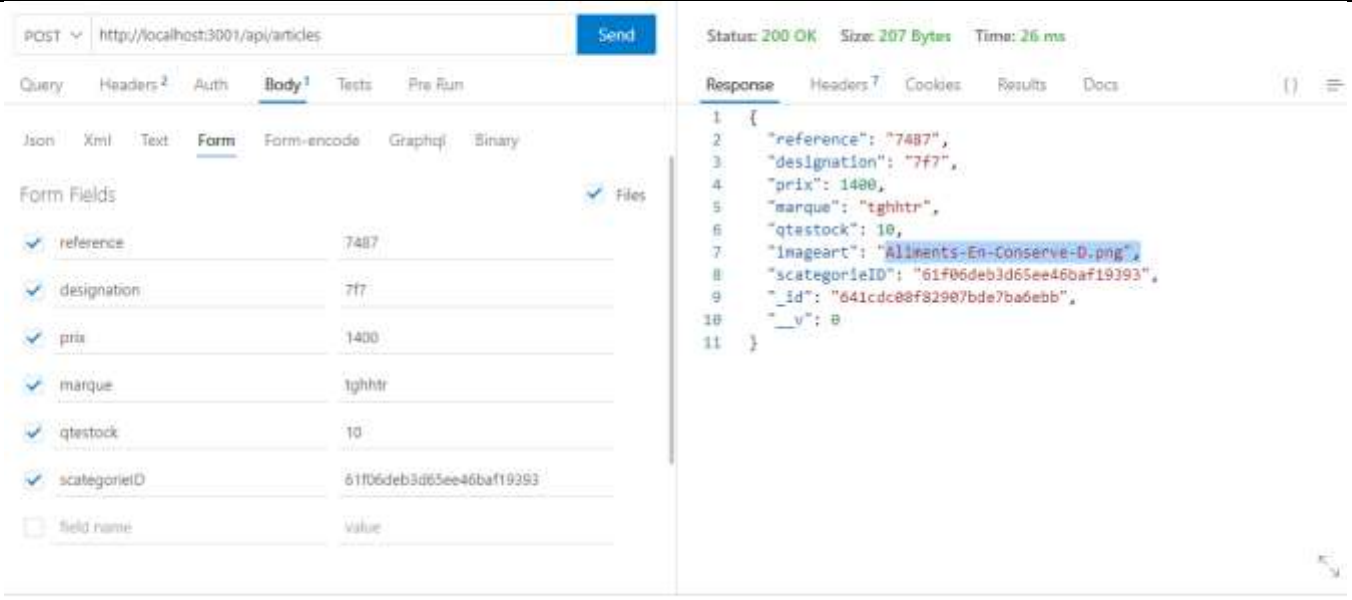
```
const  multer =require( "multer")
const fs =require( "fs")

var DIR = './images/';
if (!fs.existsSync(DIR)) {  // CREATE DIRECTORY IF NOT FOUND
  fs.mkdirSync(DIR, { recursive: true });
}
 const storage = multer.diskStorage({
  destination: (req, file, callback) => {

    callback(null, DIR);
  },
  filename: (req, file, callback) => {
    const name = file.originalname.split(' ').join('_');
    callback(null, name);
  }

});

const uploadFile = multer({
  storage: storage
});
module.exports={uploadFile};
```

Ajouter dans le fichier app.js

```
app.use(express.static(__dirname + '/'));
```

Dans le fichier article.route.js

```
const { uploadFile } = require('../middleware/upload-file')

router.post('/', uploadFile.single("imageart"),async (req, res) =>  {
    const {reference,designation,prix,marque,qtestock,scategorieID} =
req.body
    const imageart = req.file.filename
    const nouvarticle = new
Article({reference:reference,designation:designation,prix:prix,marque:marque
,qtestock:qtestock,scategorieID:scategorieID,imageart:imageart})

    try {
        await nouvarticle.save();

        res.status(200).json(nouvarticle );
    } catch (error) {
        res.status(404).json({ message: error.message });
    }


});
// chercher un article
router.get('/:articleId',async(req, res)=>{
    try {
        const art = await Article.findById(req.params.articleId);

        res.status(200).json(art);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
});
```



POST http://localhost:3001/api/articles

Status: 200 OK Size: 207 Bytes Time: 26 ms

Form Fields

| | |
|---|---|
| ✓ reference | 7487 |
| ✓ designation | 7f7 |
| ✓ prix | 1400 |
| ✓ marque | tghhtr |
| ✓ qtestock | 10 |
| ✓ scategorieID | 61f06deb3d65ee46baf19393 |

```
{
  "reference": "7487",
  "designation": "7f7",
  "prix": 1400,
  "marque": "tghhtr",
  "qtestock": 10,
  "imageart": "Aliments-En-Conserve-D.png",
  "scategorieID": "61f06deb3d65ee46baf19393",
  "_id": "641cdc08f82907bde7ba6ebb",
  "__v": 0
}
```

On peut dans une route faire appel a plusieurs middlewares

```
router.post('/', verifyToken,uploadFile.single("imageart"),async (req, res) =>  {
```

**Activer compte user**

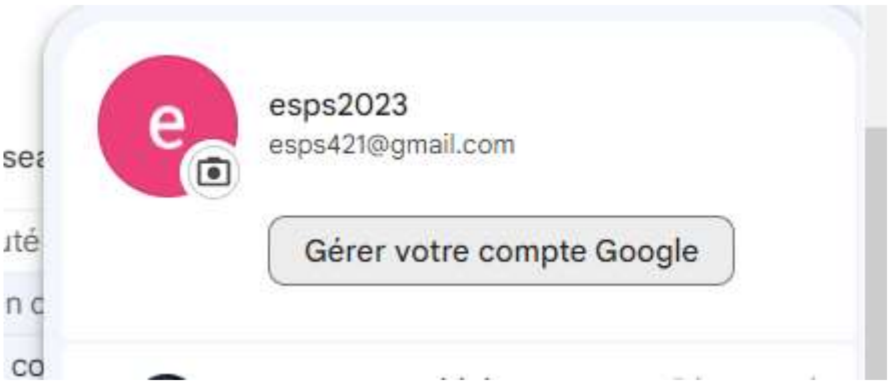
**On va utiliser nodemailer**

**npm install nodemailer**

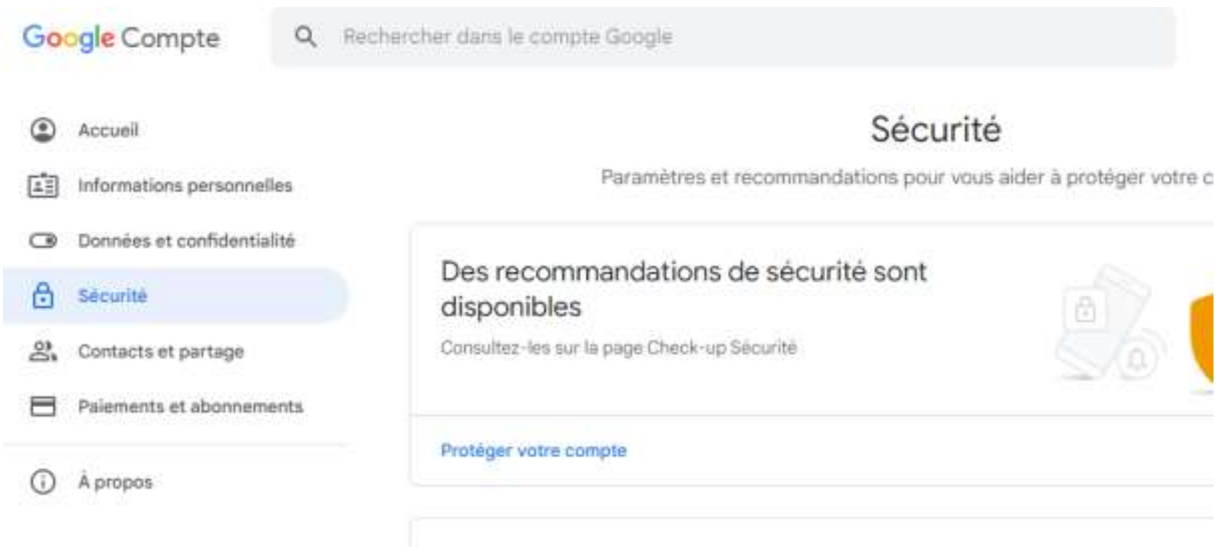
fichier backend user.route.js

```
const express = require('express');
const router = express.Router();
const User = require('../models/user.js');
const  jwt  = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const nodemailer=require('nodemailer');
const { uploadFile } = require('../middleware/uploadfile')
var transporter =nodemailer.createTransport({
  service:'gmail',

  auth:{
    user:'esps421@gmail.com',
    pass:'XXX'
  },
  tls:{
    rejectUnauthorized:false
  }
})
require('dotenv').config()
//Register

router.post('/register', uploadFile.single("avatar"),async (req, res) =>  {
  try {

         let { email, password, firstname, lastname } = req.body
         const avatar=req.file.filename
         const user = await User.findOne({ email })
         if (user) return res.status(404).send({ success: false, message: "User already 
exists" })

         const newUser = new User({ email, password, firstname, lastname,avatar })

         const createdUser = await newUser.save()

// Envoyer l'e-mail de confirmation de l'inscription
var mailOption ={
  from: '"verify your email " <esps421@gmail.com>',
  to: newUser.email,
  subject: 'vérification your email ',
  html:`<h2>${newUser.firstname}! thank you for registreting on our website</h2>
  <h4>please verify your email to procced.. </h4>
  <a 
href="http://${req.headers.host}/api/users/status/edit?email=${newUser.email}">click 
here</a>`
}
transporter.sendMail(mailOption,function(error,info){
  if(error){
    console.log(error)
  }
  else{
    console.log('verification email sent to your gmail account ')
  }
})


return res.status(201).send({ success: true, message: "Account created successfully", 
user: createdUser })


     } catch (err) {
         console.log(err)
         res.status(404).send({ success: false, message: err })

     }

 });
 * as an admin i can disable or enable an account
 */
  router.get('/status/edit/',  async (req, res) =>  {
    try {

        let  email  = req.query.email
        console.log(email)
        let user = await User.findOne({email})
        user.isActive = !user.isActive
        user.save()
        res.status(200).send({ success: true, user })
    } catch (err) {
        return res.status(404).send({ success: false, message: err })
    }
  })

  module.exports = router;
```

pour avoir le mot de passe


esps2023
esps421@gmail.com

Choisir sécurite


Google Compte
Rechercher dans le compte Google
Accueil
Informations personnelles
Données et confidentialité
Sécurité
Contacts et partage
Paiements et abonnements
À propos

Sécurité

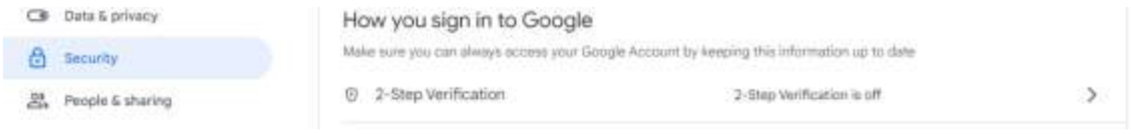
Paramètres et recommandations pour vous aider à protéger votre c

Des recommandations de sécurité sont disponibles

Consultez-les sur la page Check-up Sécurité

Protéger votre compte

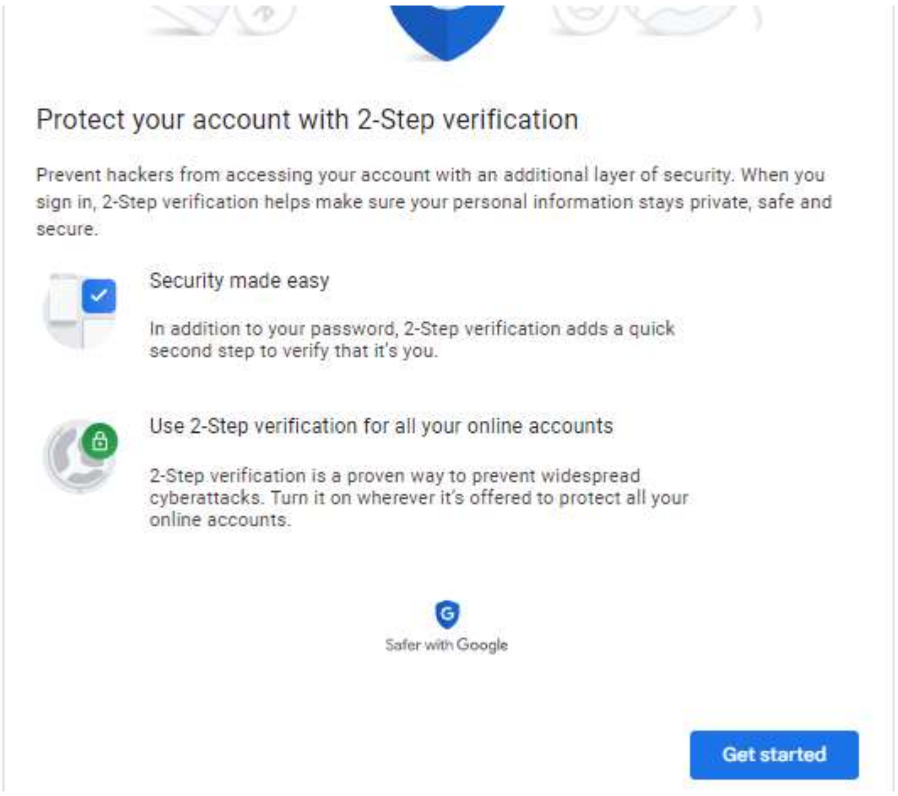
Choisir l'option 2-step verification



How you sign in to Google

Make sure you can always access your Google Account by keeping this information up to date

2-Step Verification — 2-Step Verification is off



Protect your account with 2-Step verification

Prevent hackers from accessing your account with an additional layer of security. When you sign in, 2-Step verification helps make sure your personal information stays private, safe and secure.

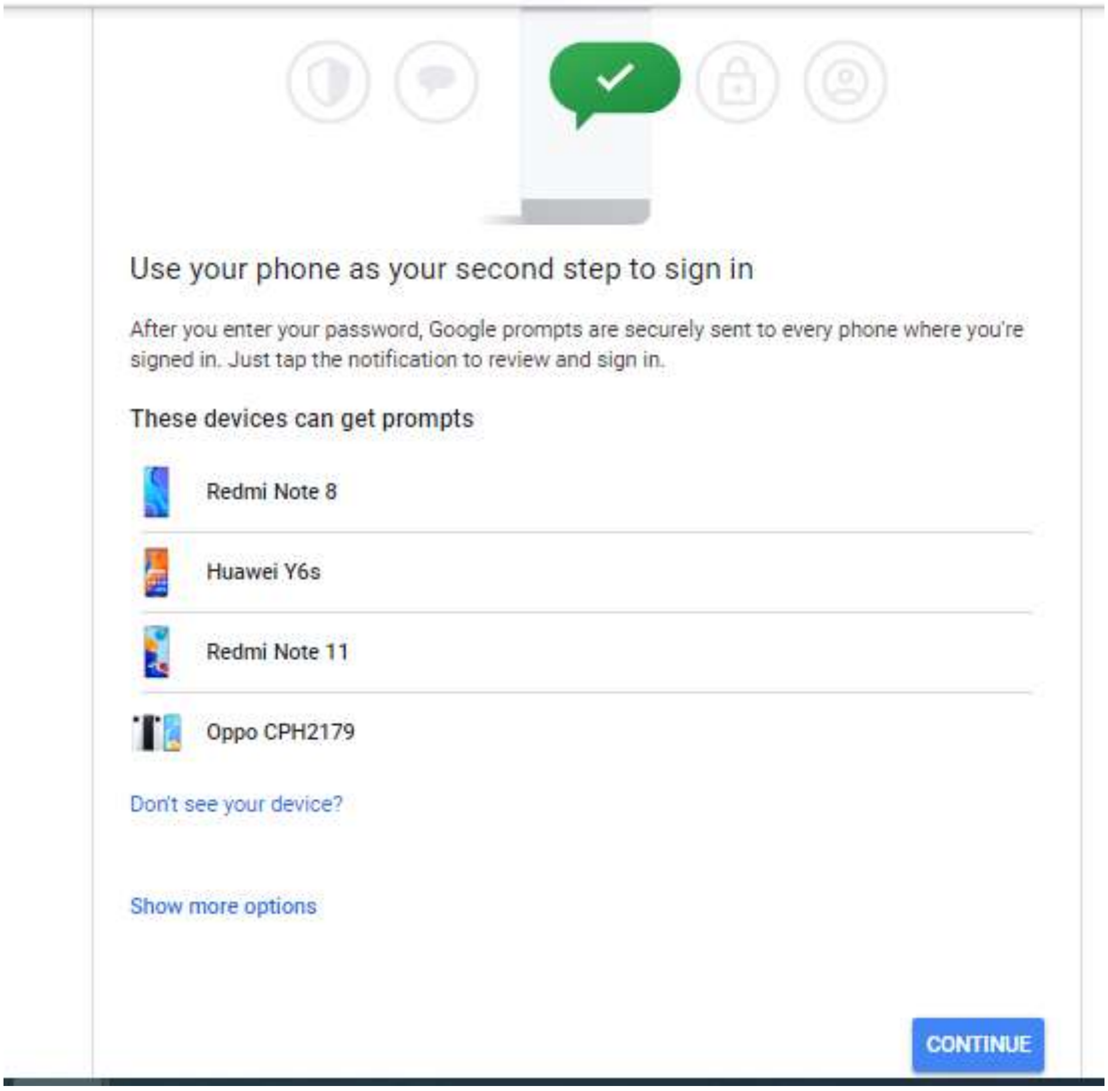
Security made easy

In addition to your password, 2-Step verification adds a quick second step to verify that it's you.

Use 2-Step verification for all your online accounts

2-Step verification is a proven way to prevent widespread cyberattacks. Turn it on wherever it's offered to protect all your online accounts.

Choisir get Started



Use your phone as your second step to sign in

After you enter your password, Google prompts are securely sent to every phone where you're signed in. Just tap the notification to review and sign in.

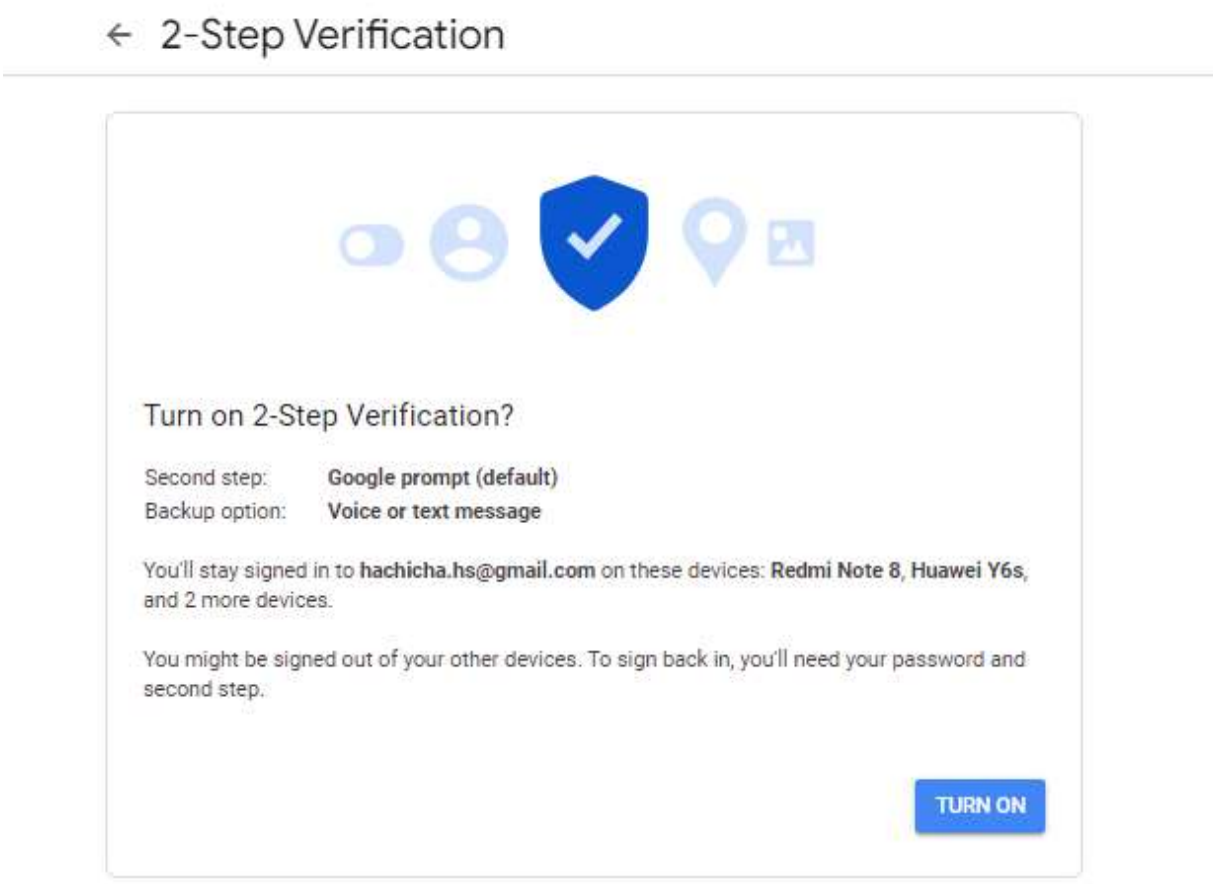
These devices can get prompts

- Redmi Note 8
- Huawei Y6s
- Redmi Note 11
- Oppo CPH2179

Don't see your device?

Show more options

Appuyer sur continue


← 2-Step Verification

Turn on 2-Step Verification?

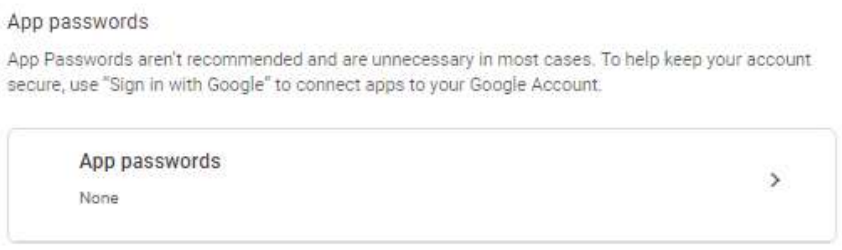
Second step: **Google prompt (default)**
Backup option: **Voice or text message**

You'll stay signed in to **hachicha.hs@gmail.com** on these devices: **Redmi Note 8**, **Huawei Y6s**, and 2 more devices.

You might be signed out of your other devices. To sign back in, you'll need your password and second step.

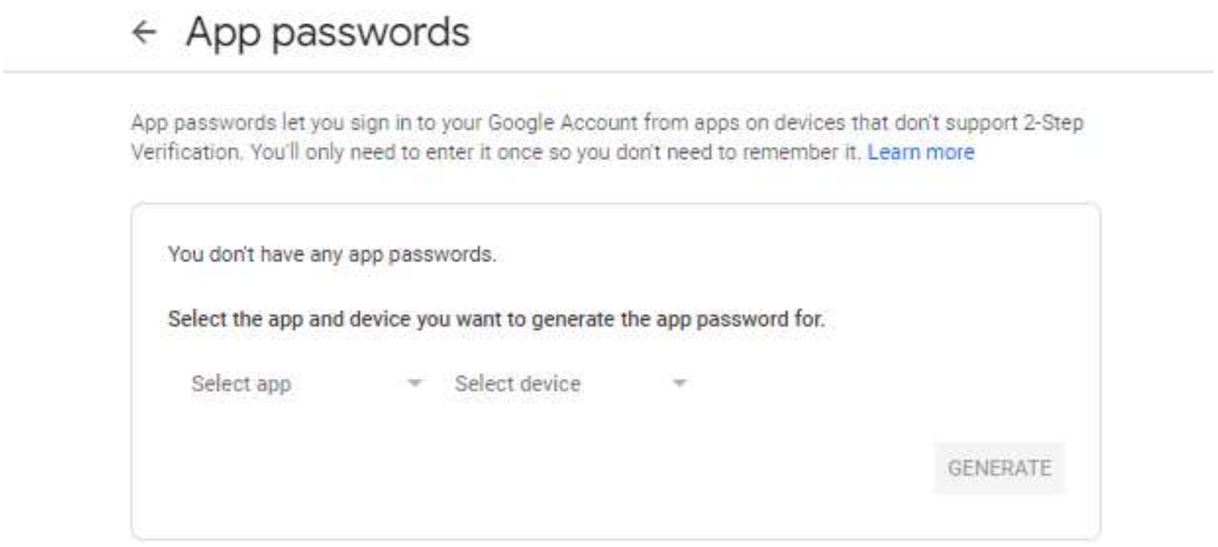
Appuyer sur turn ON

Choisir App password


App passwords

App Passwords aren't recommended and are unnecessary in most cases. To help keep your account secure, use "Sign in with Google" to connect apps to your Google Account.

App passwords
None


App passwords

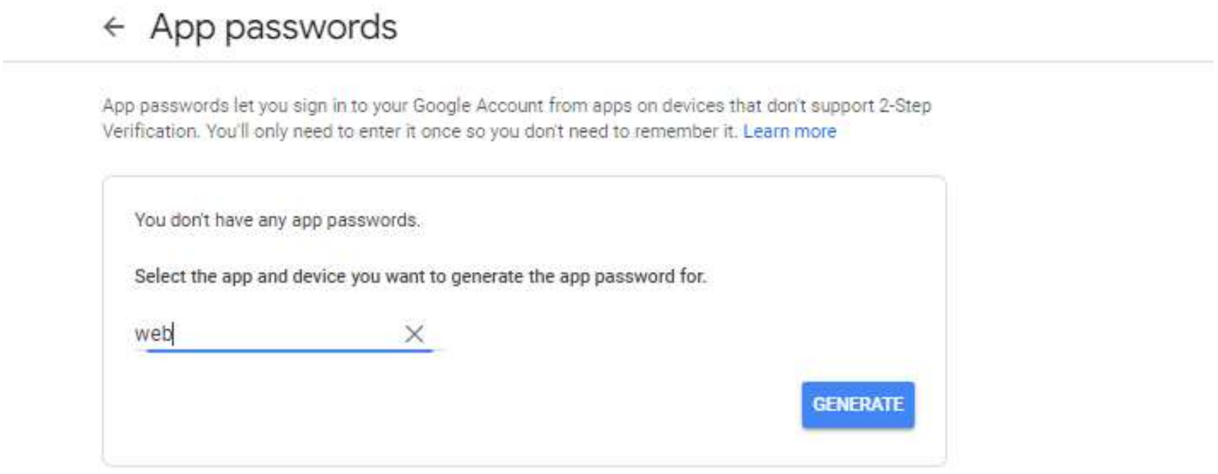
App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you don't need to remember it. Learn more

You don't have any app passwords.

Select the app and device you want to generate the app password for.

Selectionner select app

Choisir web


App passwords

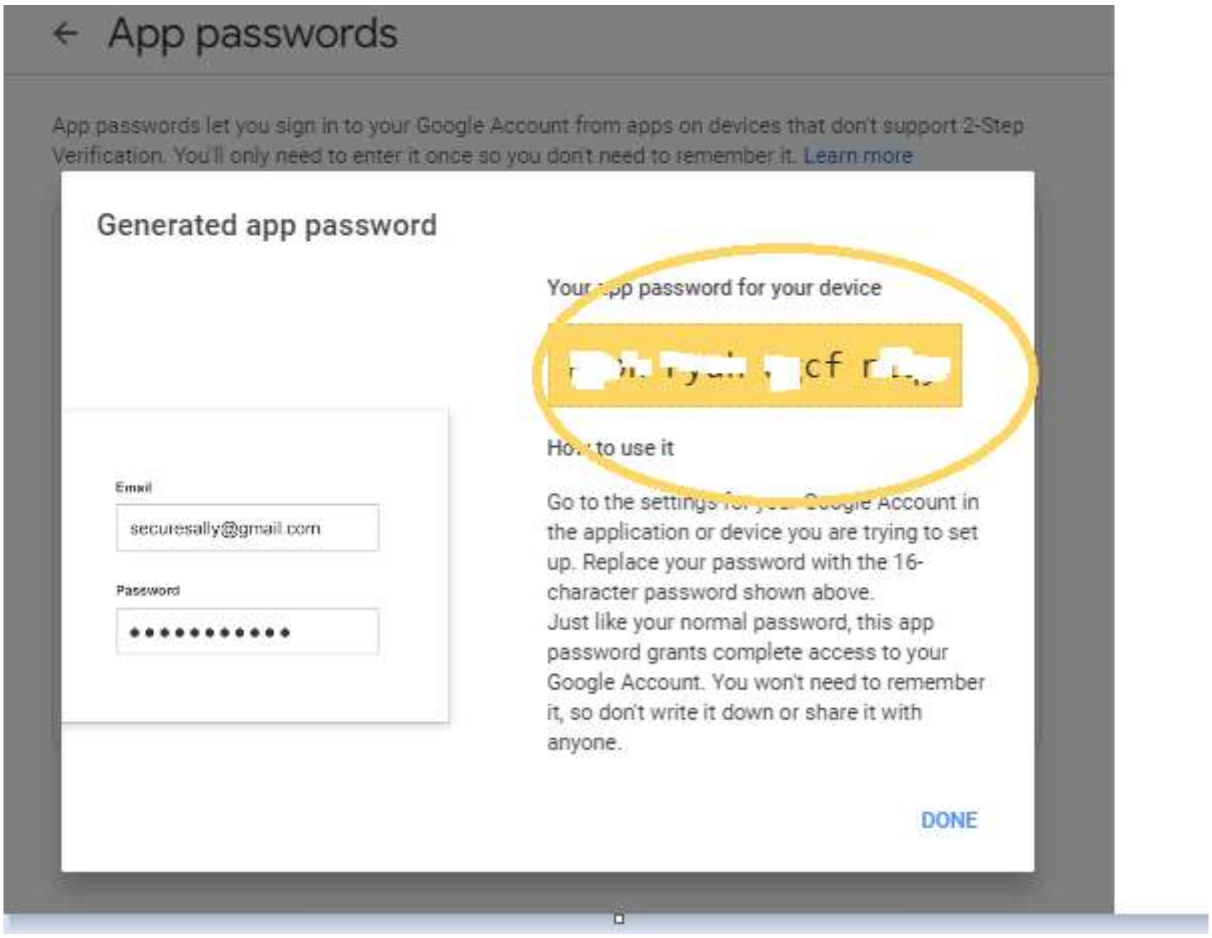
App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you don't need to remember it. Learn more

You don't have any app passwords.

Select the app and device you want to generate the app password for.

web

Appuyer sur generate


App passwords

App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you dont need to remember it. Learn more

Generated app password

Email
securesally@gmail.com

Password
●●●●●●●●●●●●

Your app password for your device

[redacted]

How to use it

Go to the settings for your Google Account in the application or device you are trying to set up. Replace your password with the 16-character password shown above.
Just like your normal password, this app password grants complete access to your Google Account. You won't need to remember it, so don't write it down or share it with anyone.

DONE

Copier le code your app password for your device

Appuyer sur done

Copier le code

```
var transporter =nodemailer.createTransport({
  service:'gmail',

  auth:{
    user:'esps421@gmail.com',
    pass:'XXXX'
  },
  tls:{
    rejectUnauthorized:false
  }
})
```

```
Messagerie

var sid  = "AC8485cd06c5ba2894a6446b60599e9021"

var auth_token ="87f4f60114312c52a3156671932c3c9c"


var twilio = require("twilio")(sid,auth_token)

exports.acceptApointments = async(req,res) =>{

try {

  console.log(req.params.id)

  const updateBook = await
book.findOneAndUpdate({Patient:req.params.id},{appointmentStatus:"accepted"}
,{new:true}).exec()

  res.json(updateBook)

  twilio.messages.create({

    from:"+19893598756",

    to: "+21629802545",

    body:"your appointment has benn accepted "

  }).then((res)=> console.log('message sent
')).catch((err)=>{console.log(err)})

} catch (error) {

      console.log("zeae")


    console.log(error)

}


}
```

3- authorizeRoles
autoriser l'accés a une route selon le role affecté a l'utilisateur

créer le fichier middleware/ authorizeRoles.js

```
const authorizeRoles = (...roles) => {
    return (req, res, next) => {
        const roleArray=[...roles]
        console.log(roleArray)
        console.log(req.user)
        if(!roles.includes(req.user.role)) {
        return res.status(401).send({ success: false, message: 'non autorisé' });
        }
        next()
    }
}
module.exports = { authorizeRoles }
```

appel au middleware authorizeRoles dans le fichier articles.route.js dans la méthode router.get

```
const express = require('express');
const router = express.Router();
const Article=require("../models/article")
const {verifyToken} =require("../middleware/veriftoken")
const {authorizeRoles} = require("../middleware/authorizeRole")
// afficher la liste des articles.

router.get('/',verifyToken,authorizeRoles("user","admin","visiteur"),async (req, res
)=> {
    try {
        const articles = await Article.find().populate("scategorieID").exec();


        res.status(200).json(articles);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }

});
```

## Refresh Token

.env

```
REFRESH_TOKEN_SECRET=qwerty
```

user.route.js

## Backend

```
const express = require('express');
const router = express.Router();
const User = require('../models/user.js');
const  jwt  = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const nodemailer=require('nodemailer');
const { uploadFile } = require('../middleware/uploadfile')
var transporter =nodemailer.createTransport({
  service:'gmail',

  auth:{
    user:'esps421@gmail.com',
    pass:'lnrqjuzysshlrpem'
  },
  tls:{
    rejectUnauthorized:false
  }
})
require('dotenv').config()
//Register

router.post('/register', uploadFile.single("avatar"),async (req, res) =>  {
  try {

         let { email, password, firstname, lastname } = req.body
         const avatar=req.file.filename
         const user = await User.findOne({ email })
         if (user) return res.status(404).send({ success: false, message: "User already
exists" })

         const newUser = new User({ email, password, firstname, lastname,avatar })

         const createdUser = await newUser.save()

// Envoyer l'e-mail de confirmation de l'inscription
var mailOption ={
  from: '"verify your email " <esps421@gmail.com>',
  to: newUser.email,
  subject: 'vérification your email ',
  html:`<h2>${newUser.firstname}! thank you for registreting on our website</h2>
  <h4>please verify your email to procced.. </h4>
  <a
href="http://${req.headers.host}/api/users/status/edit?email=${newUser.email}">click
here</a>`
}
```

```
transporter.sendMail(mailOption,function(error,info){
  if(error){
    console.log(error)
  }
  else{
    console.log('verification email sent to your gmail account ')
  }
})
  //const url =`http://localhost:3000/activate/${token}`;

     return res.status(201).send({ success: true, message: "Account created 
successfully", user: createdUser })

     } catch (err) {
         console.log(err)
         res.status(404).send({ success: false, message: err })

     }

 });


// afficher la liste des utilisateurs.
router.get('/', async (req, res, )=> {
  try {
      const users = await User.find().select("-password");
      res.status(200).json(users);
  } catch (error) {
      res.status(404).json({ message: error.message });
  }

});


// se connecter
router.post('/login', async (req, res) =>  {
  try {
      let { email, password } = req.body

      if (!email || !password) {
          return res.status(404).send({ success: false, message: "All fields are 
required" })
      }

      let user = await User.findOne({ email }).select('+password').select('+isActive')


      if (!user) {
          return res.status(404).send({ success: false, message: "Account doesn't
exists" })

      } else {

    let isCorrectPassword = await bcrypt.compare(password, user.password)
     if (isCorrectPassword) {

              delete user._doc.password
              if (!user.isActive) return res.status(200).send({ success: false,
message: 'Your account is inactive, Please contact your administrator' })

              const token = generateAccessToken(user);

             const refreshToken = generateRefreshToken(user);

              return res.status(200).send({ success: true, user,token,refreshToken })

          } else {

              return res.status(404).send({ success: false, message: "Please verify
your credentials" })

          }

      }

  } catch (err) {
      return res.status(404).send({ success: false, message: err.message })
  }

 });

//Access Token
const generateAccessToken=(user) =>{
    return jwt.sign ({ iduser: user._id, role: user.role }, process.env.SECRET, {
expiresIn: '60s'})
  }

  // Refresh
function generateRefreshToken(user) {
    return jwt.sign ({ iduser: user._id, role: user.role },
process.env.REFRESH_TOKEN_SECRET, { expiresIn: '1y'})
  }

  //Refresh Route

  router.post('/refreshToken', async (req, res, )=> {
console.log(req.body.refreshToken)
```

```
 const refreshtoken = req.body.refreshToken;
   if (!refreshtoken) {
    return res.status(404).send({success: false, message: 'Token Not Found' });
       }
   else {
       jwt.verify(refreshtoken, process.env.REFRESH_TOKEN_SECRET, (err, user) => {
         if (err) {  console.log(err)
           return res.status(406).send({ success: false,message: 'Unauthorized' });
         }
         else {
          const token = generateAccessToken(user);

          const refreshToken = generateRefreshToken(user);
          console.log("token-------",token);
         res.status(200).send({success: true,
          token,
          refreshToken
        })
           }
       });
      }


 });


 /**
* as an admin i can disable or enable an account
*/
 router.get('/status/edit/',  async (req, res) =>  {
   try {

       let  email  = req.query.email
       console.log(email)
       let user = await User.findOne({email})
       user.isActive = !user.isActive
       user.save()
       res.status(200).send({ success: true, user })
   } catch (err) {
       return res.status(404).send({ success: false, message: err })
   }
  })

 module.exports = router;
```

Faire un commit

Déterminer l’URL du git repository

```
git config --get remote.origin.url
```

update git :

```
git add .

git commit -a -m "Commit new version"

git push --set-upstream origin main
```