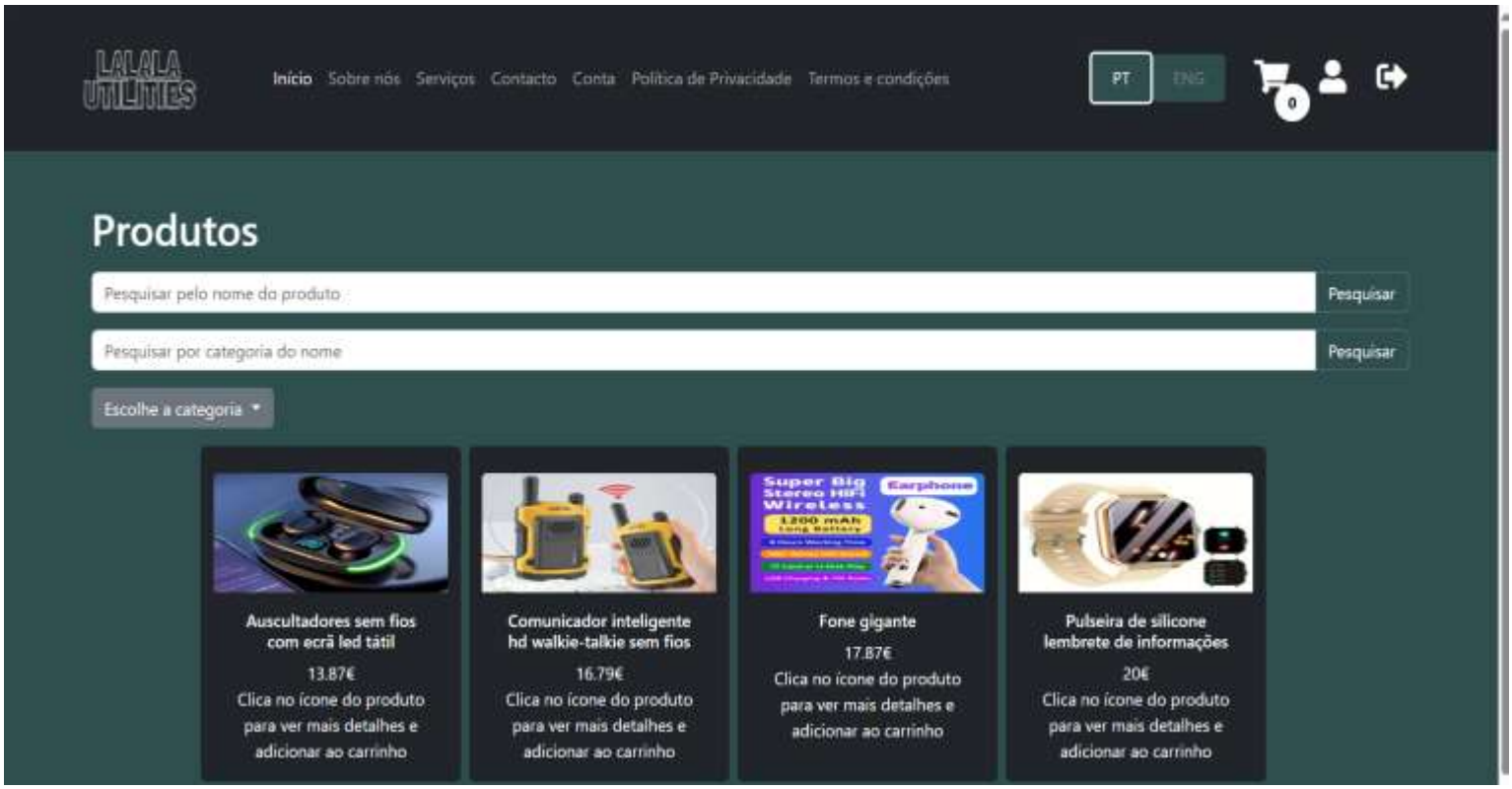
Technical Manual

Course Unit – Portfolio

2024

Online Store “Lalala Utilities”



Name: José Carlos Carranca Lúcio **Number: 202100198**

Teacher: Rui Rodrigues

Index

Content

Introduction

Objectives

This manual aims to provide comprehensive technical documentation for the online store "Lalala Utilities," developed using JavaScript, HTML, CSS, Bootstrap, Node.js, Express, jQuery, and Ajax. It serves as a detailed guide for developers and administrators, covering all the necessary aspects to understand, configure, maintain, and expand the application.

By following this manual, developers will be able to understand the structure and operation of the online store, safely make modifications and updates, and ensure that the application remains functional and efficient.

Description

General Vision

"Lalala Utilities" is an online store designed to offer a smooth and efficient shopping experience for its users. Developed using a modern technology stack, including JavaScript, Node.js, Express, jQuery, and Ajax, the application is built to be responsive, scalable, and easy to maintain. The system allows users to browse products and services, add items to the cart, make purchases, and track the status of their orders.

Purpose

The purpose of "Lalala Utilities" is to provide a robust and intuitive platform for purchasing products. The application was created to meet the needs of both end consumers and system administrators, facilitating the management of products, services, and orders.

Main Functionalities

• **Product Catalog:** Users can browse, search, and view details of the available products.
• **Shopping Cart**: Functionality to add, remove, and update the quantity of products in the cart.
• **Payment Processing:** Integration with PayPal's payment API to securely process transactions.
• **User System:** Registration, authentication, and account management for users.
• **Product Administration:** Interface for administrators to add, edit, and remove products.
• **Service Administration:** Interface for administrators to add, edit, and remove services.
• **Order Administration:** Interface for administrators to update the status of orders and review them.
• **Order History:** Users can view their order history and track the status of their orders.

Technologies Used

• **Frontend:**

- **HTML/CSS:** For the structure and styling of web pages.
- **JavaScript:** For client-side interaction logic.
- **jQuery:** For DOM manipulation and simplifying Ajax requests.

• **Backend:**

- **Node.js:** JavaScript runtime environment on the server.
- **Express:** Web framework for Node.js, used to create the server and manage routes.

• **Client-server communication:**

- **Ajax:** To make asynchronous requests and update the user interface without reloading the page.

• **Database:** Google's Firebase database was used.

Target Audience

- **End Consumers:** People interested in purchasing products online.
- **Store Administrators:** The team responsible for product management, order processing, and customer support.
- **Developers:** Professionals working on the maintenance, updates, and expansion of the application.

Application Architecture

"Lalala Utilities" follows a three-tier architecture:

1. **Presentation Layer (Frontend)**:
    - Uses HTML, CSS, Bootstrap, and JavaScript with jQuery to create a responsive and interactive user interface.
    - Ajax is used for asynchronous communication with the backend, enhancing the user experience.
2. **Application Layer (Backend):**
    - Built with Node.js and Express, this layer handles business logic, manages HTTP requests, and communicates with the database.
3. **Data Layer (Database)**:
    - Stores all essential application data, including product and service details, user information, and order records.

Performance and Scalability

"Lalala Utilities" is designed to be scalable, allowing for the addition of new features and an increasing number of users without compromising performance. The modular architecture and clear separation of responsibilities between the frontend and backend make it easier to maintain and expand the system.

Prerequisites

Development Environment

- **Node.js:** Ensure Node.js is installed. The LTS (Long Term Support) version is recommended.
    - https://nodejs.org
- **npm (Node Package Manager):** npm is usually installed alongside Node.js.

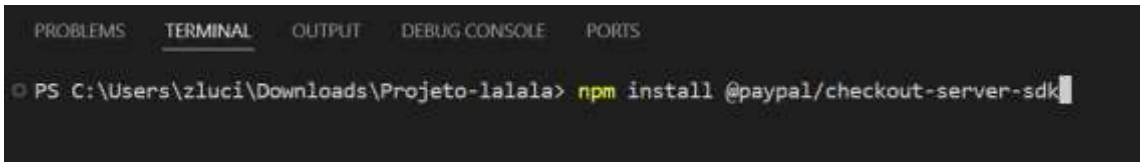
Project Dependencies

This project requires several external libraries that need to be installed.

- **@paypal/checkout-server-sdk:** Library for integrating PayPal payments.
- **axios:** HTTP client for making HTTP requests.
- **body-parser:** Middleware for parsing HTTP request bodies.
- **crypto:** Module for cryptographic functions.
- **express:** Web application framework for Node.js.
- **express-session:** Middleware for session management.
- **express-validator:** Middleware for input validation.
- **firebase:** SDK for interactions with Firebase.
- **firebase-admin:** Admin SDK for Firebase.
- **jquery:** JavaScript library for DOM manipulation and event handling.
- **multer:** Middleware for file uploads.
- **node-fetch:** Library for making HTTP requests.
- **nodemailer:** Library for sending emails.
- **oauth:** Library for OAuth 1.0.
- **oauth-1.0a:** Another library for OAuth 1.0a.

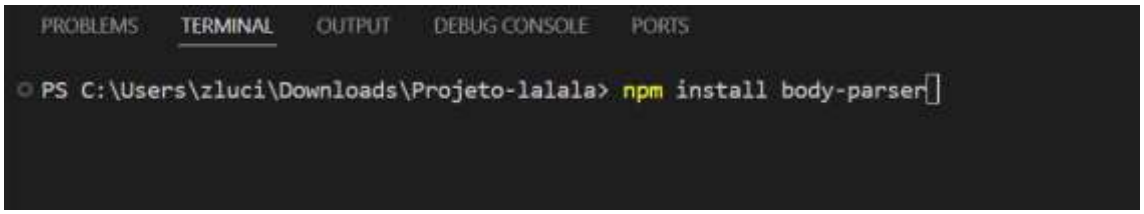
Dependency Installation

@paypal/checkout-server-sdk:

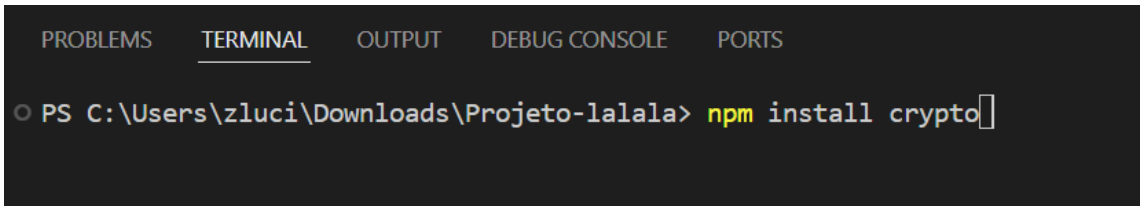


axios:

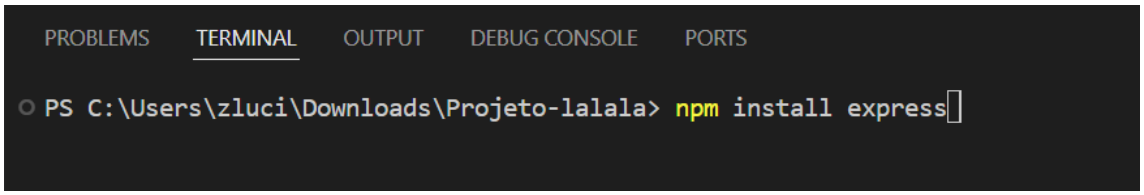
body-parser:



crypto:

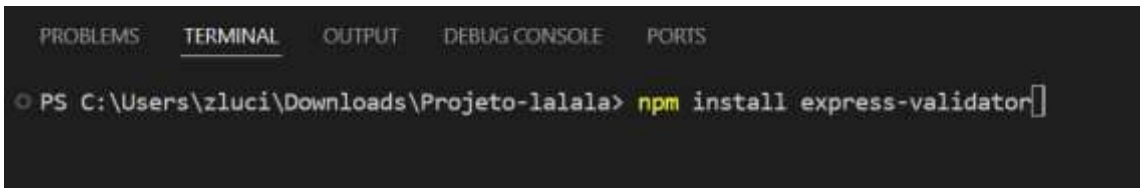


express:

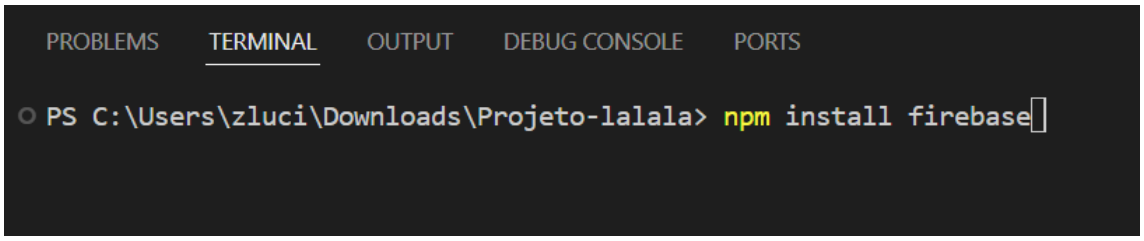


express-session:

express-validator:

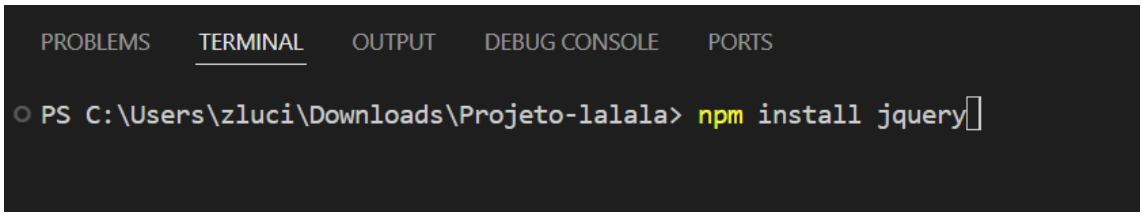


firebase:



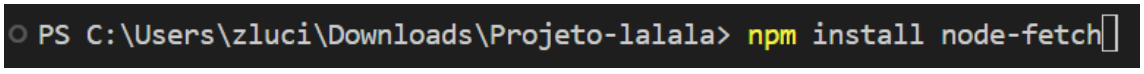
firebase-admin:

jquery:



multer:

node-fetch:



nodemailer:

```
PROBLEMS  TERMINAL  OUTPUT  DEBUG CONSOLE  PORTS
PS C:\Users\zluci\Downloads\Projeto-lalala> npm install nodemailer
```

oauth:

```
PROBLEMS  TERMINAL  OUTPUT  DEBUG CONSOLE  PORTS
PS C:\Users\zluci\Downloads\Projeto-lalala> npm install oauth
```

oauth-1.0a:

```
PROBLEMS  TERMINAL  OUTPUT  DEBUG CONSOLE  PORTS
PS C:\Users\zluci\Downloads\Projeto-lalala> npm install oauth-1.0a
```

Aditional configurations

- **@paypal/checkout-server-sdk:**

  Setting Up the Payment Environment Currently, the payment environment is set to **Sandbox** for testing purposes. The currency can be adjusted dynamically based on the user's country.

```
// Import do módulo do sdk do paypal
const paypal = require('@paypal/checkout-server-sdk');
```

```
// Cria um ambiente PayPal
const environment = new paypal.core.SandboxEnvironment(
    'YOUR_SANDBOX_ENVIRONMENT',
    'YOUR_SANDBOX_ENVIRONMENT'
);

// Cria um cliente PayPal
const client = new paypal.core.PayPalHttpClient(environment);

// Cria uma ordem de pagamento no PayPal
const request = new paypal.orders.OrdersCreateRequest();
if(totalPrice>15){
request.prefer('return=representation');

request.requestBody({
    intent: 'CAPTURE',
    purchase_units: [{
        amount: {
            currency_code: 'EUR',
            value: totalPrice.toFixed(2),
        }
    }]
});
}
```

- **firebase e firebase-admin:**
  Setting Up Firebase Database and Firebase Admin.

```
//Import do módulo 'firebase-admin'
const admin = require('firebase-admin');
//Import do módulo do sdk do firebase admin
var serviceAccount = require("Your_Json");

//Inicialização do firebase admin
var adminApp = admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
});

//export do firebase admin
module.exports = adminApp;
```

```
//Import do módulo da firebase
const firebase = require('firebase/app');

// Configuração do firebase
const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_AUTH_DOMAIN",
  projectId: "YOUR_PROJECT_ID",
  storageBucket: "YOUR_STORAGE_BUCKET",
  messagingSenderId: "YOUR_MESSAGE_SENDER_ID",
  appId: "YOUR_APP_ID"
};

// Inicializa o firebase
const firebaseApp = firebase.initializeApp(firebaseConfig);

// export do firebase
module.exports = firebaseApp;
```

- **Nodemailer:**
  Nodemailer configuration for sending emails in cases such as order status updates.

```
// Import do módulo do nodemailer
const nodemailer = require('nodemailer');
```

```
// Configura o transporte do Nodemailer utilizando o serviço Hotmail
const transporter = nodemailer.createTransport({
    service: 'hotmail',
    auth: {
        user: 'YOUR_EMAIL', // Insira seu endereço de email do Gmail
        pass: 'YOUR_PASSWORD' // Insira sua senha do Gmail
    }
});
```

```
    // Enviar email de notificação sobre a alteração do status da encomenda
    if(status !== 'Finalizada' && status !=='Cancelada'){
     mailOptions = {
        from: 'Lalalautilities2024@hotmail.com', // Endereço de email do remetente
        to: userData.email, // Endereço de email do destinatário
        subject: 'Atualização do Pedido #' + orderId, // Assunto do email
        html: `<p>Olá ${userData.name || 'Cliente'},</p>
               <p>O status do seu pedido #${orderId} foi atualizado para "${status}".</p>
               <p>Para mais detalhes, acesse a sua conta no nosso site.</p>
               <p>Cumprimentos a equipa Lalala Utilities.</p>`
    };
} else if(status === 'Finalizada'){
         mailOptions = {
        from: 'Lalalautilities2024@hotmail.com', // Endereço de email do remetente
        to: userData.email, // Endereço de email do destinatário
        subject: 'Atualização do Pedido #' + orderId, // Assunto do email
        html: `<p>Olá ${userData.name || 'Cliente'},</p>
               <p>O seu pedido #${orderId} foi Entregue. Esperemos que disfrute e volte a confiar em nós.</p>
               <p>Cumprimentos a equipa Lalala Utilities.</p>`
```

- **jQuery:**
  Place it in the HTML pages.

```
    <!-- Adicione o jQuery -->
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

- **Bootstrap:**

  Place it in HTML and CSS pages.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/5.3.0/css/bootstrap.min.css">
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.min.js"></script>
```

```
@import url("https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css");
```

- **Multer:**
  Multer facilitates the processing of files uploaded through HTTP multipart/form-data requests, commonly used in HTML forms that allow file uploads.

```
// Import do módulo multer
const multer = require("multer");
```

```
const upload = multer({
    storage: multer.memoryStorage(), // Armazenamento em memória para lidar com o buffer do arquivo
    limits: { fileSize: 10000000 }, // Limite de tamanho do arquivo em bytes
});
```

- **Express, express-session e body-parser:**

Configurations of Express Application.

```
/**importa o módulo da aplicação Express */
const app = express();

/**
 * É necessário instalar os modulos cookie-parser, express session e body-parser
 */
app.use(session({
  secret: process.env.SESSION_SECRET || 'aguardando progressões...',
  resave: false,
  saveUninitialized: true
}));

/**

/**
 * Configura o middleware body-parser para analisar requisições JSON.
 */
app.use(bodyParser.json());

/**
 * Middleware para adicionar o usuário da sessão à requisição, se houver.
 */
app.use((req, res, next) => {
    if (req.session && req.session.user) {
      // Se houver um usuário na sessão, adicione-o ao objeto de solicitação
      req.user = req.session.user;
    }
    next();
  });
```

```
// Utiliza o fs para ler o ficheiro que contém a chave privada
const privateKey = fs.readFileSync('C:/Users/zluci/Downloads/Projeto-lalala/lalala.3utilities.com/privkey.pem');

//Utiliza o fs para ler o ficheiro que contém o certificado digital
const privateCert = fs.readFileSync('C:/Users/zluci/Downloads/Projeto-lalala/lalala.3utilities.com/fullchain.pem');

// O options contém a chave e o certificado e será utilizado para criar o servidor https
const options = {
    key: privateKey,
    cert: privateCert
};

/**
 * Cria e corre o servidor https
 * @param {Object} options - As opções de certificado SSL.
 * @param {Function} callback - Função de callback a ser executada após a inicialização do servidor.
 */
https.createServer(options, app).listen(8888,function () {
    console.log('Server running at https://localhost:8888');
});

//Export da aplicação express
module.exports = app;
```

- **Crypto:**
  Used to create an encrypted code for logging in with the Twitter API.

```
// Import do módulo crypto
const crypto = require('crypto');
```

```
const codeVerifier = crypto.randomBytes(32).toString('hex');

// Calcular o code_challenge usando o code_verifier
const codeChallenge = base64url(crypto.createHash('sha256').update(codeVerifier).digest());

// Salvar o code_verifier em algum lugar para uso posterior na rota twitterCallback
req.session.codeVerifier = codeVerifier;
```

Go fetch the codeVerifier sent to req.session to verify the login permission for the store with Twitter.

```
const codeVerifier = req.session.codeVerifier;
```

Project Structure

ProjetoUC-Portfólio_202100198/

.vscode/

Settings.json

Config/

connectionFirebase-admin.js

connectionFirebase.js

lalala-3utilitites-firebase-adminsdk-t298b-8c0b4b0120.json

Lalala.3utilities.com/

Cert.pem

Chain.pem

Fullchain.pem

Privkey.pem

node_modules/

The modules of Node.js

routes/

WWW/

Images/

Email.png

Facebook.png

Google.png

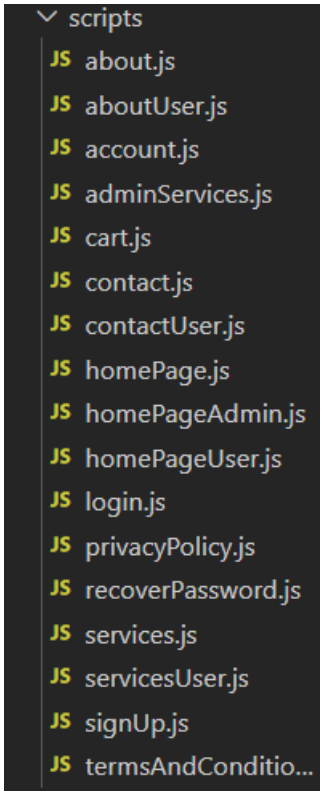
Instagram.png

Lalala logo.jpg

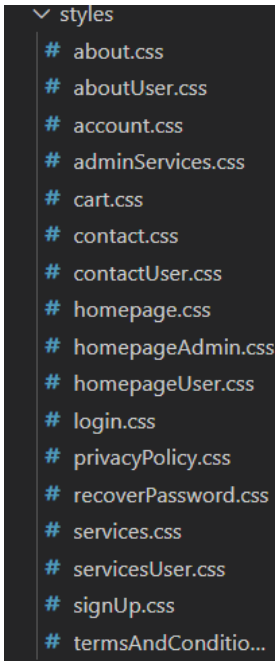
Telegram.png

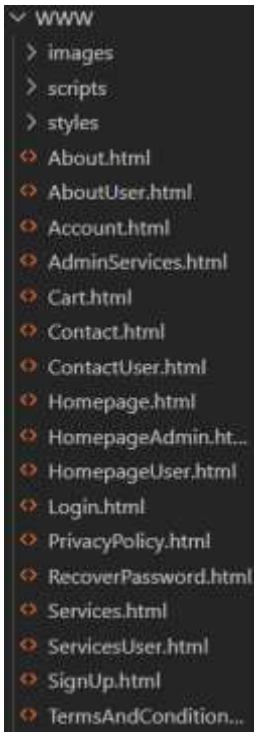
x.jpeg

Scripts/



Styles/





app.js

package-lock.json

package.json

Server Node.js e Express

Server-side configurations

The "Lalala Utilities" application server is configured using the Express framework, which is a web framework for Node.js, with HTTPS support to ensure secure communications in the app.js file.

```
/**importa o módulo da aplicação Express */
const app = express();

/**
 * É necessário instalar os modulos cookie-parser, express session e body-parser
 */
app.use(session({
  secret: process.env.SESSION_SECRET || 'aguardando progressões...',
  resave: false,
  saveUninitialized: true
}));

/**
 * Configura o middleware para servir arquivos estáticos.
 */
app.use(express.static(path.join(__dirname, 'www')));
```

```
/**
 * Configura o middleware body-parser para analisar requisições URL-encoded.
 */
app.use(bodyParser.urlencoded({ extended: true }));

/**
 * Configura o middleware body-parser para analisar requisições JSON.
 */
app.use(bodyParser.json());

/**
 * Middleware para adicionar o usuário da sessão à requisição, se houver.
 */
app.use((req, res, next) => {
    if (req.session && req.session.user) {
      // Se houver um usuário na sessão, adicione-o ao objeto de solicitação
      req.user = req.session.user;
    }
    next();
  });

//Middleware para a rota com '/' na 'homePage'
app.use('/',homePage);
```

```
// Utiliza o fs para ler o ficheiro que contém a chave privada
const privateKey = fs.readFileSync('C:/Users/zluci/Downloads/Projeto-lalala/lalala.3utilities.com/privkey.pem');

//Utiliza o fs para ler o ficheiro que contém o certificado digital
const privateCert = fs.readFileSync('C:/Users/zluci/Downloads/Projeto-lalala/lalala.3utilities.com/fullchain.pem');

// O options contém a chave e o certificado e será utilizado para criar o servidor https
const options = {
    key: privateKey,
    cert: privateCert
};

/**
 * Cria e corre o servidor https
 * @param {Object} options - As opções de certificado SSL.
 * @param {Function} callback - Função de callback a ser executada após a inicialização do servidor.
 */
https.createServer(options, app).listen(8888,function () {
    console.log('Server running at https://localhost:8888');
});
```

These configurations ensure that the "Lalala Utilities" application runs securely and efficiently, using sessions to maintain user information and serving the static files required for the user interface.

Routes

The routes created are defined in the app.js, and here are some examples.

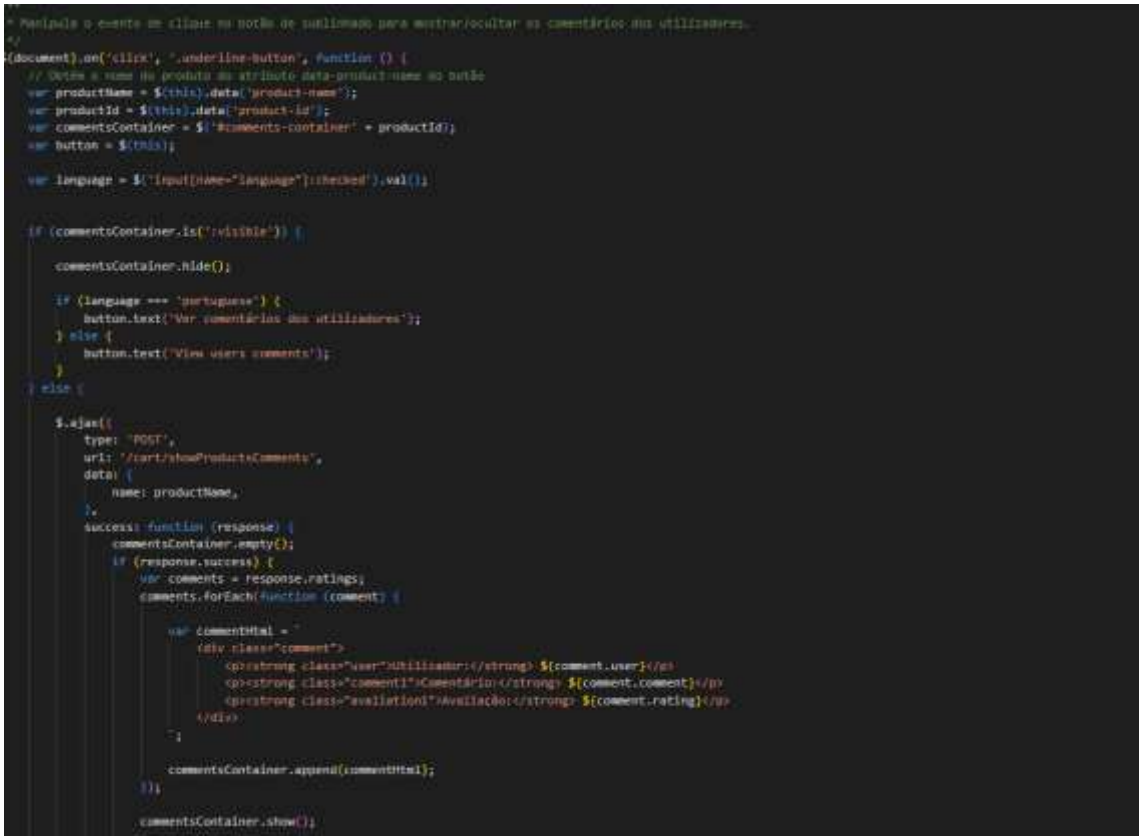
```
/**
 * Obter a referência à Home Page do user
 * @type {Object}
 */
var homePageUser = require('./routes/homePageUser.js');
```

```
router.get('/homePageUser/getProductsToSell', async function(req, res) {
    try {
        const db = getFirestore(firebaseApp);
        const productsCollection = collection(db, 'products');
        const snapshot = await getDocs(productsCollection);

        const products = [];
        snapshot.forEach(doc => {
            products.push(doc.data());
        });

        return res.status(200).json({ success: true, products: products });
    } catch (error) {
        console.error('Erro ao obter produtos:', error);
        return res.status(500).json({ success: false, message: 'Erro ao obter produtos.', error: error.message });
    }
});
```

```
router.get('/homePageUser/searchProductsByName', async (req, res) => {
    const productName = req.query.productName;
    try {
        const db = getFirestore(firebaseApp);
        const productsCollection = collection(db, 'products');
        let querySnapshot = await getDocs(productsCollection);
        const products = [];

        querySnapshot.forEach((doc) => {
            const productData = doc.data();
            if(productData.lowerCaseName.includes(productName.toLowerCase())){
                products.push(doc.data());
            }
        });

        return res.status(200).json({ success: true, products: products });
    } catch (error) {
        console.error('Erro ao pesquisar produtos:', error);
        return res.status(500).json({ success: false, message: 'Erro ao pesquisar produtos.', error: error.message });
    }
});
```

Frontend Integration

The integration with the frontend occurs through DOM manipulation, in this case using Ajax and jQuery to retrieve certain information and display it, such as products, services, and user data.



```
/**
 * Manipula o evento de clique no botão de sublinhado para mostrar/ocultar os comentários dos utilizadores.
 */
$(document).on('click', '.underline-button', function () {
    // Obtém o nome do produto do atributo data-product-name do botão
    var productName = $(this).data('product-name');
    var productId = $(this).data('product-id');
    var commentsContainer = $('#comments-container' + productId);
    var button = $(this);

    var language = $('input[name="language"]:checked').val();


    if (commentsContainer.is(':visible')) {

        commentsContainer.hide();

        if (language === 'portuguese') {
            button.text('Ver comentários dos utilizadores');
        } else {
            button.text('View users comments');
        }
    } else {

        $.ajax({
            type: 'POST',
            url: '/cart/showProductsComments',
            data: {
                name: productName,
            },
            success: function (response) {
                commentsContainer.empty();
                if (response.success) {
                    var comments = response.ratings;
                    comments.forEach(function (comment) {

                        var commentHtml = `
                            <div class="comment">
                                <p><strong class="user">Utilizador:</strong> ${comment.user}</p>
                                <p><strong class="comment1">Comentário:</strong> ${comment.comment}</p>
                                <p><strong class="avaliation1">Avaliação:</strong> ${comment.rating}</p>
                            </div>
                        `;

                        commentsContainer.append(commentHtml);
                    });

                    commentsContainer.show();
```

```
                    commentsContainer.show();

                    // Altera o texto do botão para "Esconder comentários dos utilizadores" e atualiza consoante a linguagem os campos
                    if (language === 'portuguese') {
                        button.text('Esconder comentários dos utilizadores');
                        $('.user').text('Utilizador:');
                        $('.comment1').text('Comentário:');
                        $('.avaliation1').text('Avaliação:');
                    } else {
                        button.text('Hide users comments');
                        $('.user').text('User:');
                        $('.comment1').text('Comment:');
                        $('.avaliation1').text('Avaliation:');
                    }
                }
            },
            error: function (xhr, status, error) {
                // Em caso de erro, mostre uma mensagem de erro
                modal1.show();
                document.getElementById('text2').innerText = 'Este produto não tem comentários';
            }
        });
    }
});
```

```
/**
 * Manipula o evento de carregamento do DOM para verificar o tipo de utilizador logado(admin ou utilizador comum).
 */
document.addEventListener('DOMContentLoaded', async function() {
    $.ajax({
        type: 'GET',
        url: '/contactUser/admin',
        dataType: 'json',
        success: function(response) {
            if (response.success) {
                if (response.message === 'Admin') {
                    $('#homeLink1').show();
                    $('#homeLink').hide();
                    $('#accountLink').hide();
                    $('#adminServicesLink').show();
                } else if(response.message==='User'){
                    $('#homeLink1').hide();
                    $('#homeLink').show();
                    $('#accountLink').show();
                    $('#adminServicesLink').hide();
                }
            } else {
                console.error('Erro na requisição AJAX:', response.message);
                alert(response.message);
            }
        },
        error: function(xhr, status, error) {
            console.error('Erro na requisição AJAX:', error);
            alert(response.message);
        }
    });
});
```

Main Functionalities

"Lalala Utilities" offers various features to enhance the user experience and simplify the management of the online store.

User Registration

Description: Allows new users to register in the store.

How to access: Click on the Login button and then go to Create Account.

Technical details:

- **Route:** POST /signUpForm
- **Data validation:** Checks if the email is already in use, validates the email format, and verifies the strength of the password.

```
router.post('/signUpForm', validator, async (req, res) => {
    const db = getFirestore(firebaseApp);
    try {
        // Verifica se há erros de validação
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).json({ success: false, message: 'Erro no validator', errors: errors.array() });
        }

        const { email, password, phoneNumber, userName, name } = req.body;

        // Obter o token reCAPTCHA do corpo da solicitação
        const recaptchaToken = req.body['g-recaptcha-response'];
        if (!recaptchaToken) {
            return res.status(400).json({ success: false, message: 'Token reCAPTCHA não fornecido.' });
        }

        // Verificar o token reCAPTCHA
        const secretKey = 'YOUR_RECAPTCHA_SECRET_KEY'; // Substitua pelo sua chave secreta do reCAPTCHA
        const verificationUrl = `https://www.google.com/recaptcha/api/siteverify?secret=${secretKey}&response=${reca

        const response = await axios.post(verificationUrl);
        const data = await response.data;

        // Verifica se a resposta do reCAPTCHA é válida
        if (!data.success || data.score < 0.7) {
            return res.status(400).json({ success: false, message: 'Falha na verificação reCAPTCHA. Demasiadas tenta
```

```
        // Verifica se o utilizador foi criado com sucesso
        if (userCredential && userCredential.user) {
            const user = userCredential.user;
            const userData = {
                uid: user.uid,
                userType: 'User',
                name: name,
                email: email,
                userName: userName,
                phoneNumber: phoneNumber,
                country: '',
                postalCode:'',
                region:'',
                city:'',
                address:'',
                address1:'',
                termsAndPrivacyPolitics:'Aceito',
                createdAt: new Date(),
                emailVerified: false,
            };


            // Adiciona os dados do utilizador ao Firestore
            const mycollection = collection(db, 'users');
            await setDoc(doc(mycollection, user.uid), userData);

            await sendEmailVerification(user);

            return res.status(200).json({ success: true, message: 'Usuário criado com sucesso' });
        }
    } catch (error) {
        return res.status(500).json({ success: false, message: 'Este Email já tem conta!' });
    }
);
```

Login

Description: Allow the Login of the users in store.

How to access: Click in login button and insert the credential.

Technical Details:

- **Route:** POST /loginForm
- **Data validation:** Checks if credentials match a user.

```
router.post('/loginForm', validator, async (req, res) => {
  const { email, password } = req.body;
  const auth = getAuth(firebaseApp);
  const db = getFirestore(firebaseApp);
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    res.status(500).json({ success: false, message: 'Erro ao validar o formulário' });
  } else {
    try {

      const recaptchaToken = req.body['g-recaptcha-response'];
      if (!recaptchaToken) {
        return res.status(400).json({ success: false, message: 'Token reCAPTCHA não fornecido.' });
      }

      // Verifica o token reCAPTCHA
      const secretKey = 'YOUR_RECAPTCHA_SECRET_KEY'; // Substitua pelo sua chave secreta do reCAPTCHA
      const verificationUrl = `https://www.google.com/recaptcha/api/siteverify?secret=${secretKey}&response=${recapt

      const response = await axios.post(verificationUrl);
      const data = await response.data;

      // Verifica se a resposta do reCAPTCHA é válida
      if (!data.success || data.score < 0.7) {
        return res.status(400).json({ success: false, message: 'Falha na verificação reCAPTCHA.' });
      }

      const userCredential = await signInWithEmailAndPassword(auth, email, password);
      const user = userCredential.user;
```

```
      if ((user && emailVerified) || (user && user.emailVerified)) {
        await updateDoc(doc(db, 'users', user.uid), {
          emailVerified: true
        });

        if (userDoc.exists()) {
          const userType = userDoc.data().userType;
          if (userType === 'User' || userType === 'Admin') {
            // Armazene os dados do utilizador na sessão do Express
            req.session.user = {
              uid: user.uid,
              email: user.email,
              userType: userType,
              emailVerified:user.emailVerified,
            };
            res.status(200).json({ success: true, message: `Login de ${userType.toLowerCase()} executado com sucesso` });
          } else {
            res.status(500).json({ success: false, message: 'Tipo de usuário desconhecido' });
          }
        } else {
          res.status(500).json({ success: false, message: 'Documento do usuário não encontrado' });
        }
      } else {
        res.status(500).json({ success: false, message: 'Credenciais incorretas' });
      }
    } catch (error) {
      res.status(500).json({ success: false, message: 'Erro ao validar o email' });
    }
  }
});
```

Password recovery

Description: Allow the users of the store to recover the password.

How to Access: Clicar no botão de Login e clicar em recuperar password.

Technical details:

- **Route:** POST /recoverPasswordForm
- **Data validation:** Checks if email matches a user.

```
router.post('/recoverPasswordForm', validator, async (req, res) => {
  const { email } = req.body;
  const auth = getAuth(firebaseApp);
  const errors = validationResult(req);

  try {
    if (!errors.isEmpty()) {
      res.status(500).json({ success: false, message: 'Erro ao validar o email' });
    } else {
      // Verifica se o email está associado a uma conta de utilizador
      const userRecord = await adminApp.auth().getUserByEmail(email);
      // Se o email estiver associado a uma conta de utilizador, envia o email de redefinição de senha
      if (userRecord) {
        await sendPasswordResetEmail(auth, email);
        // Retorna um JSON indicando que o email foi enviado com sucesso
        res.json({ success: true, message: 'Email enviado com sucesso' });
      } else {
        console.error('Erro ao recuperar métodos de login para o e-mail:', error);
        // Se ocorrer um erro, retorna um JSON indicando que ocorreu um erro ao enviar o email
        res.status(500).json({ success: false, message: 'Erro ao enviar o email de recuperação de senha' });
      }
    }
  } catch (error) {
    console.error('Erro ao recuperar métodos de login para o e-mail:', error);
    // Se ocorrer um erro, retorna um JSON indicando que ocorreu um erro ao enviar o email
    res.status(500).json({ success: false, message: 'Erro ao enviar o email de recuperação de senha' });
  }
});
```

Login with google

Description: Allow the register of new users or do the login in the store via Google's API.

How to access: Click in login button and after click on Google's icon.

Technical details:

- **Route:** POST /loginWithGoogle

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

    const authUrl = client.generateAuthUrl({
      access_type: 'offline', // Permite obter tokens de atualização para acesso contínuo
      scope: ['email', 'profile']
    });

    // Enviar a URL de autenticação como resposta
    res.send({ redirectUrl: authUrl });

  } catch (error) {
    console.error('Erro ao iniciar o login com o Google:', error);
    res.status(500).send('Erro ao iniciar o login com o Google');
  }
});
```

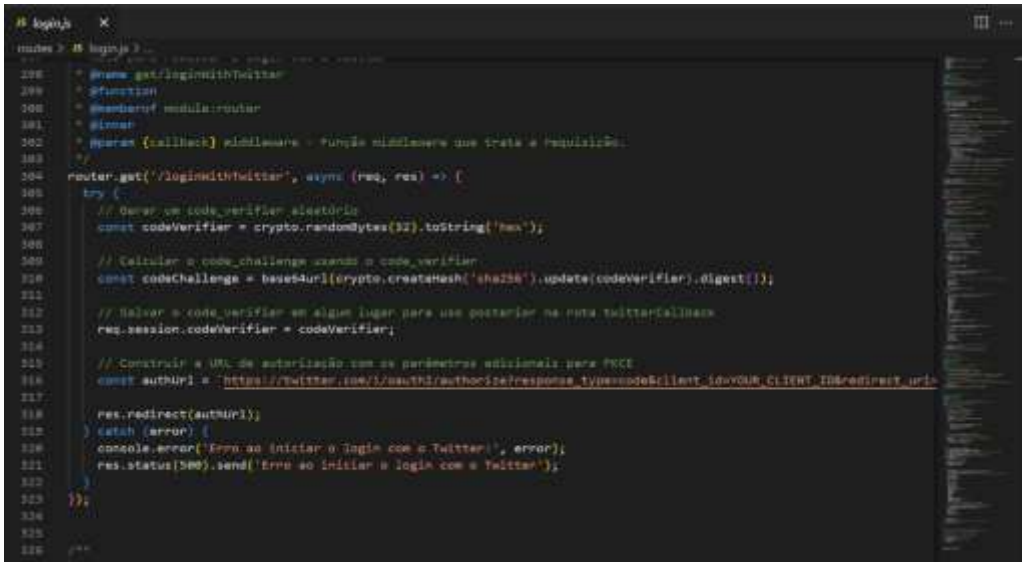
Login with twitter

Description: Allow the register of new users or do the login via Twitter's API.

How to access: Click in login button and after click on Twitter's icon.

Technical Details:

- **Route:** GET /loginWithTwitter



Search Products for name

Description: Allow the search for product name in the store.

How to Access: In search Bar put the name or the character's u want to search for.

Technical Details:

- **Route:** GET /homePage/searchProductsByName

```
router.get('/homePage/searchProductsByName', async (req, res) => {
    const productName = req.query.productName;

    try {
        const db = getFirestore(firebaseApp);
        const productsCollection = collection(db, 'products');
        let querySnapshot = await getDocs(productsCollection);
        const products = [];

        querySnapshot.forEach((doc) => {
            const productData = doc.data();
            if(productData.lowerCaseName.includes(productName.toLowerCase())){
                products.push(doc.data());
            }
        });

        return res.status(200).json({ success: true, products: products });
    } catch (error) {
        console.error('Erro ao pesquisar produtos:', error);
        return res.status(500).json({ success: false, message: 'Erro ao pesquisar produtos.', error: error.message })
    }
});
```

Search Products for categories

Description: Allow the search for categories in the store.

How to access: In Search Bar put the category or the character's u want to search.

Technical Details:

Route: GET /homePage/searchProductsByCategory

```
router.get('/homePage/searchProductsByCategory', async (req, res) => {
    const productCategory = req.query.productCategory;

    try {
        const db = getFirestore(firebaseApp);
        const productsCollection = collection(db, 'products');
        let querySnapshot = await getDocs(productsCollection);
        const products = [];

        querySnapshot.forEach((doc) => {
            const productData = doc.data();
            if(productData.categoryLowerCase.includes(productCategory.toLowerCase())){
                products.push(doc.data());
            }
        });

        return res.status(200).json({ success: true, products: products });
    } catch (error) {
        console.error('Erro ao pesquisar produtos:', error);
        return res.status(500).json({ success: false, message: 'Erro ao pesquisar produtos.', error: error.message })
    }
});
```

Search Products for Drop Down Menu

Description: Allow the search for categories or subcategories on store.

How to access: In Drop Down Menu select the category u want.

Technical Details:

Route: GET /homePage/searchProductsByCategory

```
router.get('/homePage/getProductsByCategory', async (req, res) => {
    const productCategory = req.query.productCategory;
    const formattedProductCategory = capitalizeFirstLetter(productCategory);
    try {
        const db = getFirestore(firebaseApp);
        const productsCollection = collection(db, 'products');

        let querySnapshot;
        const products = [];

        if (!productCategory) {
            // Se productCategory estiver vazio, executa a consulta sem filtros
            querySnapshot = await getDocs(productsCollection);

        } else {
            // Caso contrário, pesquisa pela categoria do produto
            const q = query(productsCollection, where('category', '==', formattedProductCategory));
            querySnapshot = await getDocs(q);
        }


        querySnapshot.forEach((doc) => {
            products.push(doc.data());
        });

        return res.status(200).json({ success: true, products: products });
    } catch (error) {
        console.error('Erro ao pesquisar produtos:', error);
        return res.status(500).json({ success: false, message: 'Erro ao pesquisar produtos.', error: error.message });
    }
});
```

```
router.get('/homePage/getProductsBySubCategory', async (req, res) => {
    const productSubCategory = req.query.productSubCategory;
    const formattedProductSubCategory = productSubCategory.toLowerCase();
    try {
        const db = getFirestore(firebaseApp);
        const productsCollection = collection(db, 'products');

        let querySnapshot;
        const products = [];

        if (!productSubCategory) {
            // Se productSubCategory estiver vazio, executa a consulta sem filtros
            querySnapshot = await getDocs(productsCollection);

        } else {
            // Caso contrário, pesquisa pela sub categoria do produto
            const q = query(productsCollection, where('subCategory', '==', formattedProductSubCategory));
            querySnapshot = await getDocs(q);
        }


        querySnapshot.forEach((doc) => {
            products.push(doc.data());
        });

        return res.status(200).json({ success: true, products: products });
    } catch (error) {
        console.error('Erro ao pesquisar produtos:', error);
        return res.status(500).json({ success: false, message: 'Erro ao pesquisar produtos.', error: error.message });
    }
});
```

Place page content in English or Portuguese

Description: Allow place all the content on the pages in English or Portuguese.

How to Access: In Button “PT” put in Portuguese and in the button “ENG” put in English.

```
function changeLanguage(language) {
    if (language === 'pt') {
        document.getElementById('homeLink').innerText = 'Início';
        document.getElementById('aboutLink').innerText = 'Sobre';
        document.getElementById('servicesLink').innerText = 'Serviços';
        document.getElementById('contactLink').innerText = 'Contacto';
        document.getElementById('Login').innerText = 'Entrar';
        document.getElementById('privacyPoliticsLink').innerText = 'Política de privacidade';
        $('.cardText').text('Faz o registo/login para adicionares o produto ao carrinho');
        $('#categoryDropdownMenuButton').text('Escolha a categoria');
        $('#subCategoryDropdownMenuButton').text('Escolha a subcategoria');
        $('.cardText1').text('Clica no icon para ver mais detalhes');
        $('#products').text('Produtos');
        $('#searchButton').text('Pesquisar');
        document.getElementById('searchButton1').innerText = 'Pesquisar';
        $('#searchInput').attr('placeholder', 'Pesquisar por nome de produto');
        $('#searchInput1').attr('placeholder', 'Pesquisar por categoria de produto');
        $('#customize').text('Customiza o teu produto ->');
        $('#contactUs').text('Contacta-nos');
        document.getElementById('termsAndConditionsLink').innerText = 'Termos e condições';
        $('.colors1').text('Cores:');
        $('.sizes1').text('Tamanhos:');
        $('.productDescription').text('Descrição:');
        $('.productPrice').text('Preço:');
        $('.closeModal').text('Fechar');
        $('.mb-2.commentsOfUsers').text('Comentários dos utilizadores');
        $('.underline-button').text('Ver comentários dos utilizadores');
        $('.user').text('Utilizador:');
        $('.comment1').text('Comentário:');
        $('.avaliation1').text('Avaliação:');
        $('.avaliation').text('Avaliação');
    } else {
```

```
    } else {
        document.getElementById('homeLink').innerText = 'Home';
        document.getElementById('aboutLink').innerText = 'About';
        document.getElementById('privacyPoliticsLink').innerText = 'Privacy Politics';
        document.getElementById('servicesLink').innerText = 'Services';
        document.getElementById('contactLink').innerText = 'Contact';
        document.getElementById('Login').innerText = 'Login';
        $('.cardText').text('Sign in to can add the product to cart');
        $('#categoryDropdownMenuButton').text('Choose the category');
        $('#subCategoryDropdownMenuButton').text('Choose the subcategory');
        $('.cardText1').text('Click on icon to see more details');
        $('#products').text('Products');
        $('#searchButton').text('Search');
        document.getElementById('searchButton1').innerText = 'Search';
        $('#searchInput').attr('placeholder', 'Search for product name');
        $('#searchInput1').attr('placeholder', 'Search for product category');
        $('#customize').text('Customize ur product ->');
        $('#contactUs').text('Contact Us');
        document.getElementById('termsAndConditionsLink').innerText = 'Terms and Conditions';
        $('.colors1').text('Colors:');
        $('.sizes1').text('Sizes:');
        $('.productDescription').text('Description:');
        $('.productPrice').text('Price:');
        $('.closeModal').text('Close');
        $('.mb-2.commentsOfUsers').text('Comments of Users');
        $('.underline-button').text('View users comments');
        $('.user').text('User:');
        $('.comment1').text('Comment:');
        $('.avaliation1').text('Avaliation:');
        $('.avaliation').text('Avaliation');
    }
}
```

Search Services for name

Description: Allow the search for products by name.

How to access: In Search Bar put the name u want to search or the character's.

Technical Details:

- **Route:** GET /services/searchServicesByName

```
router.get('/services/searchServicesByName', async (req, res) => {
    const serviceName = req.query.serviceName;

    try {
        const db = getFirestore(firebaseApp);
        const servicesCollection = collection(db, 'Services');
        const querySnapshot = await getDocs(servicesCollection);

        const services = [];

        querySnapshot.forEach(doc => {
            const serviceData = doc.data();
            // Verifique se o nome do serviço contém a parte pesquisada
            if (serviceData.lowerCaseName.includes(serviceName.toLowerCase())) {
                services.push(serviceData);
            }
        });

        return res.status(200).json({ success: true, services: services });
    } catch (error) {
        console.error('Erro ao pesquisar serviços:', error);
        return res.status(500).json({ success: false, message: 'Erro ao pesquisar serviços.', error: error.message });
    }
});
```

Obter dados da conta no carrinho

Description: Allows you to obtain user account data to use in the cart payment process.

How to access: On the cart page, click on the get account details button.

Technical details:

- **Route:** GET /getData

```
router.get('/getData', async function (req, res) {
    try {
        const user = req.session.user;
        if (user) {
            const db = getFirestore(firebaseApp);
            const userUid = user.uid;
            const userDoc = await getDoc(doc(db, 'users', userUid));

            // Extrair os dados do documento do utilizador
            const userData = userDoc.data();

            // Enviar os dados do utilizador como resposta
            res.status(200).json({ success: true, userData: userData });
        } else {
            res.status(400).json({ success: false, message: 'Usuário não autenticado' });
        }
    } catch (error) {
        console.error('Erro ao obter dados do usuário:', error);
        res.status(500).json({ success: false, message: 'Erro ao processar a solicitação' });
    }
});
```

Payment for products

Description: Allows payment for products in the store.

How to access: On the cart page, using the paypal buttons, choose the payment method.

Technical details:

- **Route:** POST /create-order

```
router.post('/create-order', async function (req, res) {
    try {
        // Verifica se o utilizador está autenticado
        const currentUser = req.session.user;
        if (!currentUser) {
            return res.status(401).json({ success: false, message: 'Usuário não autenticado.' });
        }

        const db = getFirestore(firebaseApp);
        const userDocRef = doc(db, 'users', currentUser.uid);
        const userDocSnapshot = await getDoc(userDocRef);

        if (!userDocSnapshot.exists() || !userDocSnapshot.data().email || !userDocSnapshot.data().emailVerified) {
            return res.status(400).json({ success: false, message: 'O usuário não tem um email válido.' });
        }

        // Calcula o valor total do carrinho
        let totalAmount = 0;
        let shipping = 0;

        const userCartRef = doc(db, 'cart', currentUser.uid);
        const productsCollectionRef = collection(userCartRef, 'products');
        const snapshot = await getDocs(productsCollectionRef);

        const cartProducts = [];


        snapshot.forEach(doc => {
            totalAmount += parseFloat(doc.data().price.replace(',', '.'));
            shipping += parseFloat(doc.data().shipping.replace(',', '.'));
            cartProducts.push(doc.data());
        });

        // Total do preço com base nos produtos remanescentes no carrinho
        if (cartProducts.length > 4) {
            shipping /= 2;
```

```
        if (cartProducts.length > 4) {
            shipping /= 2;
        }

        let totalPrice = totalAmount + shipping;

        // Cria um ambiente PayPal
        const environment = new paypal.core.SandboxEnvironment(
            '[illegible]',
            '[illegible]'
        );

        // Cria um cliente PayPal
        const client = new paypal.core.PayPalHttpClient(environment);

        // Cria uma ordem de pagamento no PayPal
        const request = new paypal.orders.OrdersCreateRequest();
        if(totalPrice>15){
        request.prefer('return=representation');

        request.requestBody({
            intent: 'CAPTURE',
            purchase_units: [{
                amount: {
                    currency_code: 'EUR',
                    value: totalPrice.toFixed(2),
                }
            }]
        });
    }

        // Executa a solicitação do PayPal
        const response = await client.execute(request);
        const orderID = response.result.id;
        const approveUrl = response.result.links.find(link => link.rel === 'approve').href;

        return res.status(200).json({ success: true, message: 'Pedido criado', orderID: orderID });
    } catch (error) {
        console.error('Erro ao processar o pagamento e criar pedido:', error);
        return res.status(500).json({ success: false, message: 'Erro ao processar o pagamento e criar pedido.', error: error.message });
    }
});
```

Edit username

Description: Allows you to modify the username.

How to access: On the account page, click on the edit button that belongs to the username.

Technical details:

- **Route:** POST /edit-username

```
router.post('/edit-username', async function (req, res) {
    const user = req.session.user;
    if (!user) {
        return res.status(401).json({ success: false, message: 'Usuário não autenticado.' });
    }

    try {
        const newUserName = req.body.newUserName;
        const userId = user.uid;
        const myCollection = collection(db, 'users');
        const userDocRef = doc(myCollection, userId);

        await updateDoc(userDocRef, { userName: newUserName });

        res.status(200).json({ success: true, message: 'Nome de usuário atualizado com sucesso.' });
    } catch(error) {
        res.status(500).json({ success: false, message: 'Erro ao alterar o nome de usuário.' });
    }
});
```

Edit name

Description: Allows you to modify the name.

How to access: On the account page, click on the edit button next to the name.

Technical details:

- **Route:** POST /edit-name

```
router.post('/edit-name', async function (req, res) {
    const user = req.session.user;
    if (!user) {
        return res.status(401).json({ success: false, message: 'Usuário não autenticado.' });
    }

    try {
        const newName = req.body.newName;
        const userId = user.uid;
        const myCollection = collection(db, 'users');
        const userDocRef = doc(myCollection, userId);

        await updateDoc(userDocRef, { name: newName });

        res.status(200).json({ success: true, message: 'Nome atualizado com sucesso.' });
    } catch(error) {
        res.status(500).json({ success: false, message: 'Erro ao alterar o nome.' });
    }
});
```

Edit email

Description: Allows you to modify the email.

How to access: On the account page, click on the edit button that belongs to the email.

Technical details:

- **Route:** POST /edit-email

```
router.post('/edit-email', async function (req, res) {
    const user = req.session.user;
    const newEmail = req.body.newEmail;
    if (!user) {
        return res.status(401).json({ success: false, message: 'Usuário não autenticado.' });
    }
        try {

            const userId = user.uid;
            const myCollection = collection(db, 'users');
            const userDocRef = doc(myCollection, userId);

            await updateDoc(userDocRef, {
                email: newEmail,
                emailVerified: false // Definindo emailVerified como false
            });

            await adminApp.auth().updateUser(userId,{
                email:newEmail,
            });

            const link = await adminApp.auth().generateEmailVerificationLink(newEmail);

            const mailOptions = {
                from: 'Lalalautilities2024@hotmail.com',
                to: newEmail,
                subject: 'Verificação de e-mail',
                html: `Olá,<br><br>Por favor, clique no link abaixo para verificar seu e-mail:<br><br><a href="${link
            };
            transporter.sendMail(mailOptions, function(error, info){
              if (error) {
                console.error('Erro ao enviar e-mail:', error);
                res.status(500).json({ success: false, message: 'Erro ao enviar e-mail de verificação.' });
              } else {
                console.log('E-mail enviado:', info.response);
                res.status(200).json({ success: true, message: 'E-mail de verificação enviado com sucesso.' });
              }
            });

            res.status(200).json({ success: true, message: 'Email atualizado com sucesso.' });
        } catch(error) {
            res.status(500).json({ success: false, message: 'Erro ao alterar o Email do utilizador.' });
        }
});
```

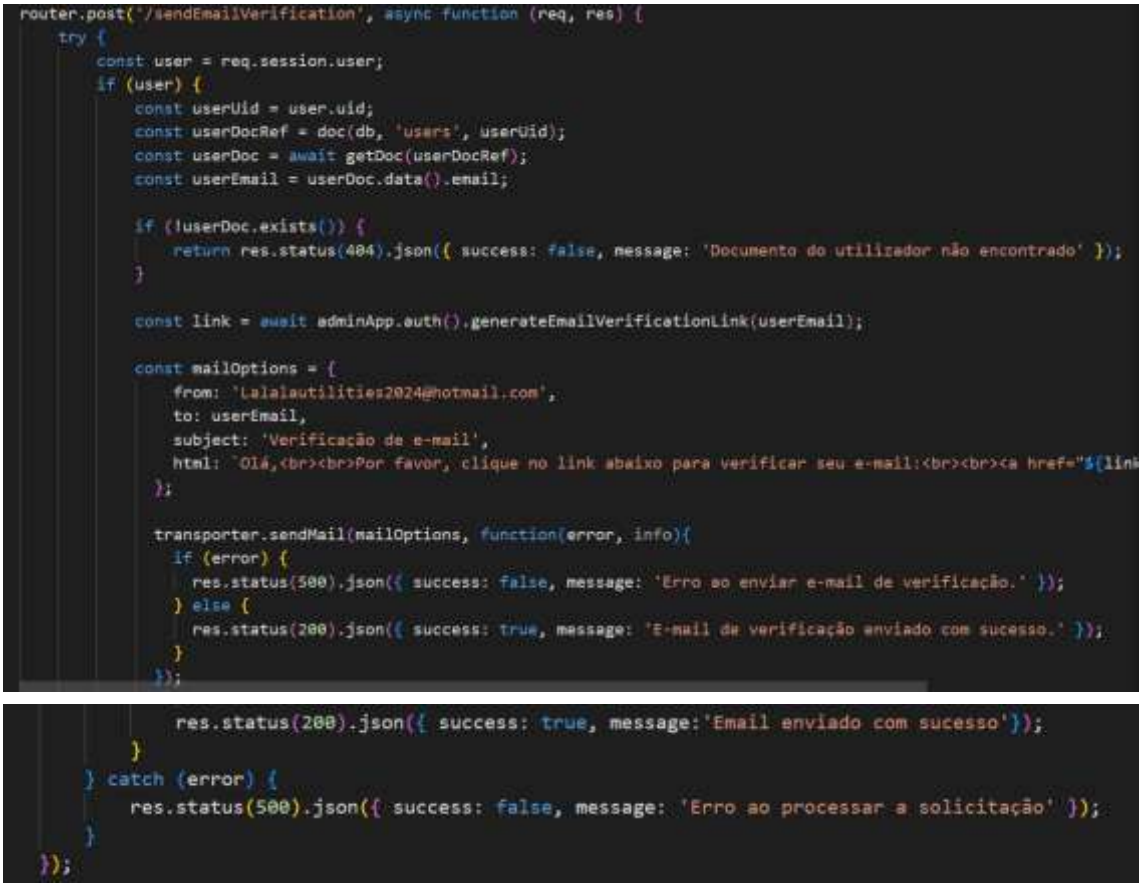
Send email verification email

Description: Allows you to check your email.

How to access: On the account page, click the check email button that belongs to the email.

Technical details:

- **Route:** POST /sendEmailVerification



```
router.post('/sendEmailVerification', async function (req, res) {
    try {
        const user = req.session.user;
        if (user) {
            const userUid = user.uid;
            const userDocRef = doc(db, 'users', userUid);
            const userDoc = await getDoc(userDocRef);
            const userEmail = userDoc.data().email;

            if (!userDoc.exists()) {
                return res.status(404).json({ success: false, message: 'Documento do utilizador não encontrado' });
            }

            const link = await adminApp.auth().generateEmailVerificationLink(userEmail);

            const mailOptions = {
                from: 'Lalalautilities2024@hotmail.com',
                to: userEmail,
                subject: 'Verificação de e-mail',
                html: `Olá,<br><br>Por favor, clique no link abaixo para verificar seu e-mail:<br><br><a href="${link
            };

            transporter.sendMail(mailOptions, function(error, info){
                if (error) {
                    res.status(500).json({ success: false, message: 'Erro ao enviar e-mail de verificação.' });
                } else {
                    res.status(200).json({ success: true, message: 'E-mail de verificação enviado com sucesso.' });
                }
            });

            res.status(200).json({ success: true, message:'Email enviado com sucesso'});
        }
    } catch (error) {
        res.status(500).json({ success: false, message: 'Erro ao processar a solicitação' });
    }
});
```

Send password change email

Description: Allows you to send an email to change your password.

How to access: On the account page, click on the edit button that belongs to password.

Technical details:

- **Route:** POST /edit-password

```
router.post('/edit-password', async function (req, res) {
    const user = req.session.user;
    if (!user) {
        return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
    }
        try {
            const currentEmail = user.email;
            const auth = getAuth(firebaseApp);
            await sendPasswordResetEmail(auth, currentEmail);

            res.status(200).json({ success: true, message: 'Email enviado com sucesso' });
        } catch(error) {
            res.status(500).json({ success: false, message: 'Erro ao enviar o email.' });
        }
});
```

Edit mobile number

Description: Allows you to change your cell phone number.

How to access: On the account page, click on the edit button that belongs to the mobile number.

Technical details:

- **Route:** POST /edit-phoneNumber

```
router.post('/edit-phoneNumber', async function (req, res) {

    const newPhoneNumber = req.body.newPhoneNumber;
    const user = req.session.user;

    if (!user) {
        return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
    }
        try {
            const userId = user.uid;
            const myCollection = collection(db, 'users');
            const userDocRef = doc(myCollection, userId);

            await updateDoc(userDocRef, { phoneNumber: newPhoneNumber });

            res.status(200).json({ success: true, message: 'Número de telemóvel atualizado com sucesso.' });
        } catch(error) {
            console.error('Erro ao alterar o nome de usuário:', error);
            res.status(500).json({ success: false, message: 'Erro ao alterar o nome de usuário.' });
        }
});
```

Edit country

Description: Allows you to change the country.

How to access: On the account page, click on the edit button that belongs to the country.

Technical details:

- **Route:** POST /edit-country

```
router.post('/edit-country', async function (req, res) {
    const user = req.session.user;
    const newCountry = req.body.newCountry;
    if (!user) {
        return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
    }
        try {
            const userId = user.uid;
            const myCollection = collection(db, 'users');
            const userDocRef = doc(myCollection, userId);

            await updateDoc(userDocRef, { country: newCountry });

             res.status(200).json({ success: true, message: 'País atualizado com sucesso.' });
        } catch(error) {
            res.status(500).json({ success: false, message: 'Erro ao alterar País.' });
        }
});
```

Edit district/region

Description: Allows changing the district/region.

How to access: On the account page, click on the edit button that belongs to the district/region.

Technical details:

- **Route:** POST /edit-region

```
router.post('/edit-region', async function (req, res) {
    const user = req.session.user;
    const newRegion = req.body.newRegion;
    if (!user) {
        return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
    }
        try {
            const userId = user.uid;
            const myCollection = collection(db, 'users');
            const userDocRef = doc(myCollection, userId);

            await updateDoc(userDocRef, { region: newRegion });

             res.status(200).json({ success: true, message: 'Região atualizada com sucesso.' });
        } catch(error) {
            res.status(500).json({ success: false, message: 'Erro ao alterar Região.' });
        }
});
```

Edit city

Description: Allows you to change the city.

How to access: On the account page, click on the edit button that belongs to the city.

Technical details:

- **Route:** POST /edit-city

```
router.post('/edit-city', async function (req, res) {
    const user = req.session.user;
    const newCity = req.body.newCity;
    if (!user) {
        return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
    }
        try {
           const userId = user.uid;
           const myCollection = collection(db, 'users');
           const userDocRef = doc(myCollection, userId);

           await updateDoc(userDocRef, { city: newCity });

            res.status(200).json({ success: true, message: 'Cidade atualizada com sucesso.' });
        } catch(error) {
            res.status(500).json({ success: false, message: 'Erro ao alterar Cidade.' });
        }
});
```

Edit address

Description: Allows you to change the Address.

How to access: On the account page, click on the edit button that belongs to the address.

Technical details:

- **Route:** POST /edit-address

```
router.post('/edit-address', async function (req, res) {
    const currentUser = req.session.user;
    const newAddress = req.body.newAddress;
    if (!currentUser) {
        return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
    }
        try {
           const userId = currentUser.uid;
           const myCollection = collection(db, 'users');
           const userDocRef = doc(myCollection, userId);

           await updateDoc(userDocRef, { address: newAddress });

            res.status(200).json({ success: true, message: 'Endereço atualizado com sucesso.' });
        } catch(error) {
            res.status(500).json({ success: false, message: 'Erro ao alterar Endereço.' });
        }
});
```

Edit address 1

Description: Allows changing of Address 1.

How to access: On the account page, click on the edit button that belongs to address 1.

Technical details:

- **Route:** POST /edit-address1

```
router.post('/edit-address1', async function (req, res) {
    const currentUser = req.session.user;
    const newAddress1 = req.body.newAddress1;
    if (!currentUser) {
        return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
    }
        try {
            const userId = currentUser.uid;
            const myCollection = collection(db, 'users');
            const userDocRef = doc(myCollection, userId);

            await updateDoc(userDocRef, { address1: newAddress1 });

             res.status(200).json({ success: true, message: 'Endereço atualizado com sucesso.' });
        } catch(error) {
            res.status(500).json({ success: false, message: 'Erro ao alterar Endereço.' });
        }
});
```

Edit zip code

Description: Allows you to change the postal code.

How to access: On the account page, click on the edit button that belongs to the postal code.

Technical details:

- **Route:** POST /edit-postalCode

```
router.post('/edit-postalCode', async function (req, res) {
    const currentUser = req.session.user;
    const newPostalCode = req.body.newPostalCode;
    if (!currentUser) {
        return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
    }
        try {
            const userId = currentUser.uid;
            const myCollection = collection(db, 'users');
            const userDocRef = doc(myCollection, userId);

            await updateDoc(userDocRef, { postalCode: newPostalCode });

             res.status(200).json({ success: true, message: 'Código Postal atualizado com sucesso.' });
        } catch(error) {
            res.status(500).json({ success: false, message: 'Erro ao alterar Código Postal.' });
        }
});
```

Log Out

Description: Allows you to exit the session.

How to access: On the account page, click on the logout button that belongs to Sign Out.

Technical details:

- **Route:** POST /logOut

```
router.post('/logOut', async function (req, res) {
    const user = req.session.user;
    if (!user) {
        return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
    }
        try {
           const auth =  getAuth(firebaseApp);
           await signOut(auth);
            req.session.destroy();
            res.status(200).json({ success: true, message: 'LogOut com sucesso.' });
        } catch(error) {
            res.status(500).json({ success: false, message: 'Erro no LogOut.' });
        }
});
```

View orders placed

Description: Allows you to see the orders you have placed.

How to access: On the account page, click on the view orders button below the orders title.

Technical details:

- **Route:** GET /getOrders

```
router.get('/getOrders', async function(req, res) {
    const currentUser = req.session.user;
    try {
        const userId = currentUser.uid;
        const db = getFirestore(firebaseApp);
        const ordersCollection = collection(db, 'orders');
        const snapshot = await getDocs(ordersCollection);

        const orders = [];
        snapshot.forEach(doc => {
            const orderUserId = doc.data().userId;
            if(orderUserId === userId){
                orders.push(doc.data());
            }
        });


        return res.status(200).json({ success: true, orders: orders,userId: userId });
    } catch (error) {
        return res.status(500).json({ success: false, message: 'Erro ao obter produtos.', error: error.message });
    }
});
```

Delete account

Description: Allows you to delete the account and all data.

How to access: On the account page, click on the trash button located next to delete account.

Technical details:

- **Route:** POST /deleteAccount

```
router.post('/deleteAccount', async function (req, res) {
    try {
        // Verifica se o utilizador está autenticado
        const currentUser = req.session.user;
        if (!currentUser) {
            return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
        }

        // Obtém uma referência para a coleção produtos no carrinho do utilizador
        const db = getFirestore(firebaseApp);
        const userCartRef = doc(db, 'cart', currentUser.uid);
        await deleteDoc(userCartRef);

        // Obtém os documentos da coleção produtos
        const userDocRef = doc(db, 'users', currentUser.uid);
        await deleteDoc(userDocRef);
        await adminApp.auth().deleteUser(currentUser.uid);

        req.session.destroy();

        // Retorna os produtos encontrados
        return res.status(200).json({ success: true, message: 'Utilizador eliminado com sucesso' });
    } catch (error) {
        // Em caso de erro, retorna uma mensagem de erro
        return res.status(500).json({ success: false, message: 'Erro ao eliminar utilizador', error: error.message });
    }
});
```

Add product to cart

Description: Allows you to add a product to the cart.

How to access: log in, and on the homepage click on the product, select the color and size and click on the add to cart button.

Technical details:

- **Route:** POST /addToCart

```
router.post('/addToCart', async (req, res) => {
    const productName = req.body.name;
    const productColor = req.body.color;
    const productSize = req.body.size;
    const formattedProductName = capitalizeFirstLetter(productName);

    try {
        const currentUser = req.session.user; // Obtém o utilizador atualmente logado

        if (!currentUser) {
            return res.status(401).json({ success: false, message: 'Usuário não autenticado.' });
        }

        const db = getFirestore(firebaseApp);
        const productsCollection = collection(db, 'products');

        // Consulta o produto pelo nome
        const q = query(productsCollection, where('name', '==', formattedProductName));
        const querySnapshot = await getDocs(q);

        // Verifica se o produto foi encontrado
        if (querySnapshot.empty) {
            return res.status(404).json({ success: false, message: 'Produto não encontrado.' });
        }

        const productDoc = querySnapshot.docs[0];
        const productData = productDoc.data();
```

```
        const productDoc = querySnapshot.docs[0];
        const productData = productDoc.data();


        const userCartRef = doc(collection(db, 'cart'), currentUser.uid);
        const productsCollectionRef = collection(userCartRef, 'products');

        // Adiciona o produto ao carrinho do utilizador
        await addDoc(productsCollectionRef, {
            ...productData,
            colors: productColor,
            size: productSize
        });

        return res.status(200).json({ success: true });
    } catch (error) {
        console.error('Erro ao adicionar o produto ao carrinho:', error);
        return res.status(500).json({ success: false, message: 'Erro ao adicionar o produto ao carrinho.' });
    }
});
```

Send feedback on purchased products when you receive the products

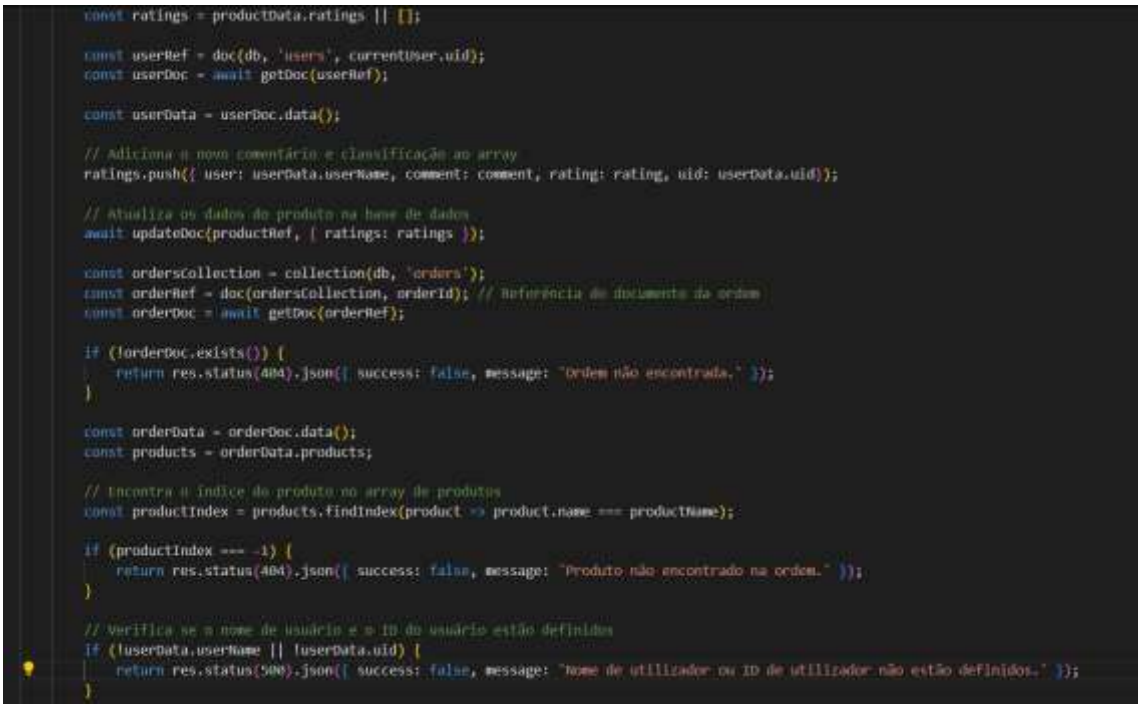
Description: Allows you to send opinions about purchased products.

How to access: On the account page, when the order has the status of completed, a button will appear to evaluate the products purchased.

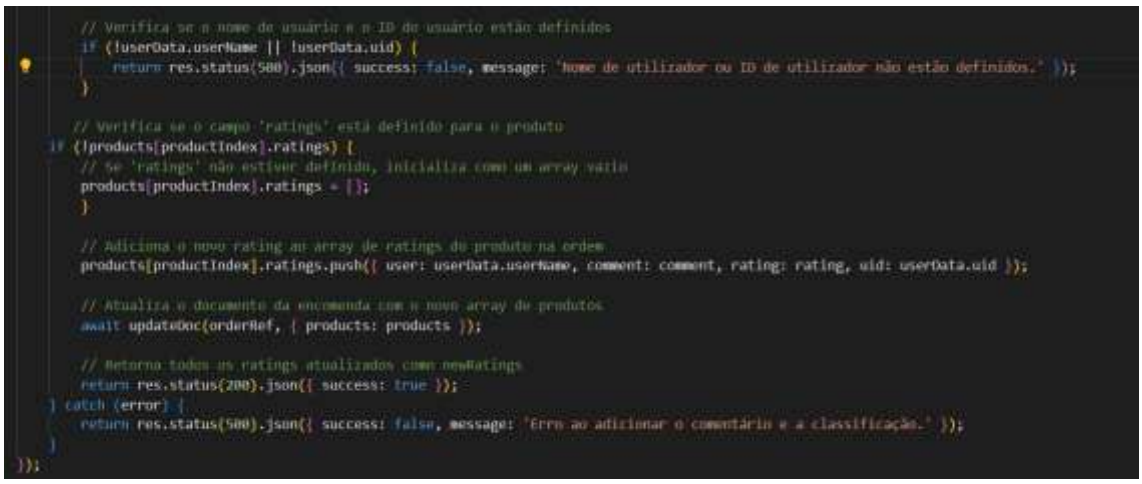
Technical details:

- **Route:** POST /sendOpinion

```
router.post('/sendOpinion', async (req, res) => {
    const productName = req.body.name;
    const comment = req.body.comment;
    const rating = req.body.rating;
    const orderId = req.body.orderID;

    try {
        const currentUser = req.session.user; // Obtém o utilizador atualmente logado

        if (!currentUser) {
            return res.status(401).json({ success: false, message: 'Utilizador não autenticado.' });
        }

        const db = getFirestore(firebaseApp);
        const productsCollection = collection(db, 'products');

        // Procura o produto pelo nome
        const querySnapshot = await getDocs(query(productsCollection, where('name', '==', productName)));

        if (querySnapshot.empty) {
            return res.status(404).json({ success: false, message: 'Produto não encontrado.' });
        }

        // Obtém o ID do produto
        const productId = querySnapshot.docs[0].id;

        // Atualiza o array de classificações do produto
        const productRef = doc(db, 'products', productId);
        const productDoc = await getDoc(productRef);

        if (!productDoc.exists()) {
            return res.status(404).json({ success: false, message: 'Produto não encontrado.' });
        }

        const productData = productDoc.data();
        const ratings = productData.ratings || [];
```



```
    const ratings = productData.ratings || [];

    const userRef = doc(db, 'users', currentUser.uid);
    const userDoc = await getDoc(userRef);

    const userData = userDoc.data();

    // Adiciona o novo comentário e classificação ao array
    ratings.push({ user: userData.userName, comment: comment, rating: rating, uid: userData.uid});

    // Atualiza os dados do produto na base de dados
    await updateDoc(productRef, { ratings: ratings });

    const ordersCollection = collection(db, 'orders');
    const orderRef = doc(ordersCollection, orderId); // Referência do documento da ordem
    const orderDoc = await getDoc(orderRef);

    if (!orderDoc.exists()) {
        return res.status(404).json({ success: false, message: "Ordem não encontrada." });
    }

    const orderData = orderDoc.data();
    const products = orderData.products;

    // Encontra o índice do produto no array de produtos
    const productIndex = products.findIndex(product => product.name === productName);

    if (productIndex === -1) {
        return res.status(404).json({ success: false, message: "Produto não encontrado na ordem." });
    }

    // Verifica se o nome de usuário e o ID do usuário estão definidos
    if (!userData.userName || !userData.uid) {
        return res.status(500).json({ success: false, message: "Nome de utilizador ou ID de utilizador não estão definidos." });
    }
```



```
    // Verifica se o nome de usuário e o ID do usuário estão definidos
    if (!userData.userName || !userData.uid) {
        return res.status(500).json({ success: false, message: 'Nome de utilizador ou ID de utilizador não estão definidos.' });
    }

    // Verifica se o campo 'ratings' está definido para o produto
    if (!products[productIndex].ratings) {
        // Se 'ratings' não estiver definido, inicializa como um array vazio
        products[productIndex].ratings = [];
    }

    // Adiciona o novo rating ao array de ratings do produto na ordem
    products[productIndex].ratings.push({ user: userData.userName, comment: comment, rating: rating, uid: userData.uid });

    // Atualiza o documento da encomenda com o novo array de produtos
    await updateDoc(orderRef, { products: products });

    // Retorna todos os ratings atualizados como newRatings
    return res.status(200).json({ success: true });
} catch (error) {
    return res.status(500).json({ success: false, message: 'Erro ao adicionar o comentário e a classificação.' });
}
});
```

Administrator Features

Add products

Description: Allows you to add products to the database.

How to access: On the admin page, select products, fill in the fields and click on the add product button.

Technical details:

- **Route:** POST /addProduct

```
router.post('/addProduct', upload.fields([
    { name: 'photo', maxCount: 1 },
    { name: 'photo1', maxCount: 1 },
    { name: 'photo2', maxCount: 1 },
    { name: 'photo3', maxCount: 1 },
    { name: 'photo4', maxCount: 1 },
    { name: 'photo5', maxCount: 1 }
]), async function (req, res) {
    const { category,subCategory,colors, size,name, description,shipping, price } = req.body;
    const formattedName = capitalizeFirstLetter(name);
    const formattedCategory = capitalizeFirstLetter(category);
    const formattedDescription = capitalizeFirstLetter(description);
    const storage = getStorage(firebaseApp);

    try {
        const photoUrls = [];

        // Função para fazer upload de uma única foto
        const uploadPhoto = async (photoFile, fileName) => {
            if (photoFile && photoFile.length > 0) {
                const formattedFileName = capitalizeFirstLetter(fileName);
                const photoRef = ref(storage, `products/${formattedCategory}/${formattedFileName}.jpg`);
                const metadata = {
                    contentType: 'image/jpeg',
                };
                await uploadBytes(photoRef, photoFile[0].buffer, metadata);
                const downloadURL = await getDownloadURL(photoRef);
                return downloadURL;
            }
            return null; // Retorna null se não houver foto
        };

        // Upload da primeira foto
        const photoUrl = await uploadPhoto(req.files['photo'], formattedName);
        if (photoUrl) {
            photoUrls.push(photoUrl);
        }
```

```
        for (let i = 1; i <= 5; i++) {
            const fieldName = `photo${i}`;
            const photoUrl = await uploadPhoto(req.files[fieldName], `${formattedName}_${i}`);
            if (photoUrl) {
                photoUrls.push(photoUrl);
            }
        }

        const newPrice = price + '€';
        const newShipping = shipping + '€';
        const sanitizedLowerCaseName = name.replace(/[^\s\wÀ-ÿ]/gi, '').toLowerCase();
        const sanitizedCategoryLowerCase = category.replace(/[^\s\wÀ-ÿ]/gi, '').toLowerCase();
        const subCategoryLowerCase = subCategory.toLowerCase();
        // Dados do produto
        const productData = {
            category: formattedCategory,
            name: formattedName,
            description: formattedDescription,
            price: newPrice,
            photos: photoUrls,
            hidden: 'No',
            subCategory:subCategoryLowerCase,
            size:size.split(',').map(prodSize => prodSize.trim()+' cm'),
            colors:colors.split(',').map(prodColor => prodColor.trim()),
            shipping:newShipping,
            lowerCaseName: sanitizedLowerCaseName,
            categoryLowerCase: sanitizedCategoryLowerCase,
        };

        // Armazenar os dados do produto no Firestore
        const db = getFirestore(firebaseApp);
        const productsCollection = collection(db, 'products');
        await setDoc(doc(productsCollection, formattedName), productData);

        return res.status(200).json({ success: true, message: 'Produto adicionado com sucesso.' });
    } catch (error) {
        return res.status(500).json({ success: false, message: 'Erro ao adicionar produto.', error: error.message });
    }
});
```

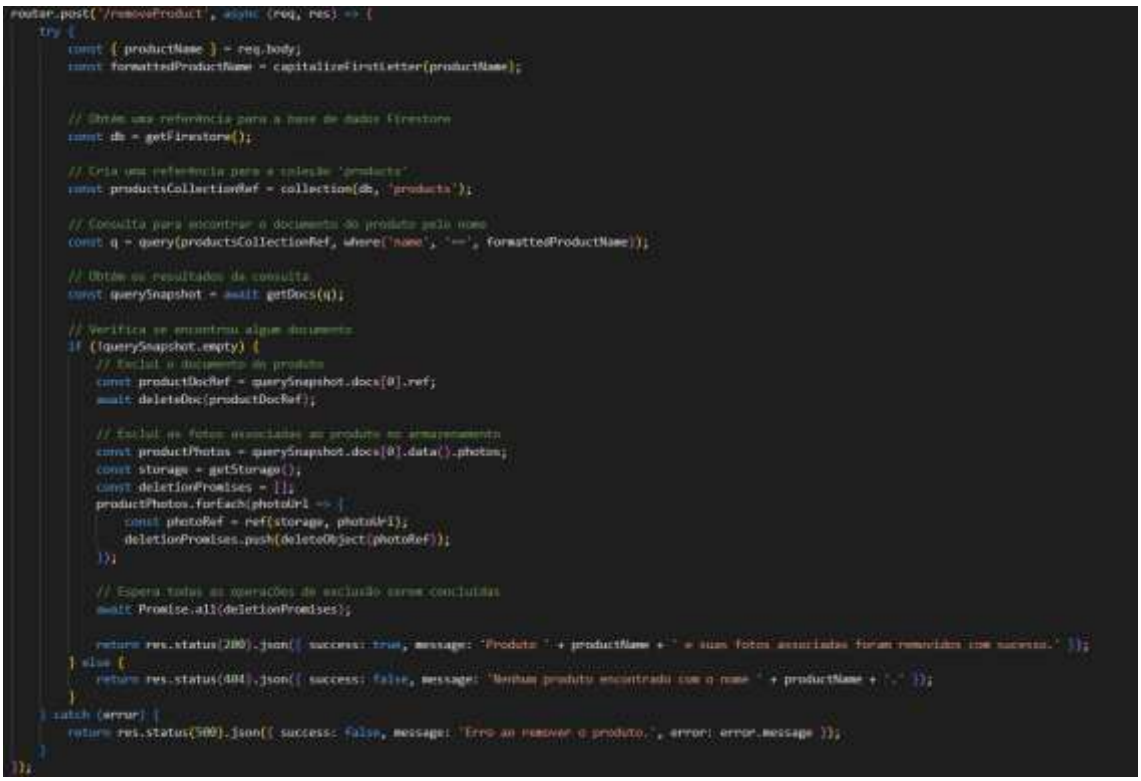
Remove products

Description: Allows you to remove products from the database.

How to access: On the admin page, select products, scroll to see the products and click on the trash icon to remove the desired product.

Technical details:

- **Route:** POST /removeProduct



```
router.post('/removeProduct', async (req, res) => {
    try {
        const { productName } = req.body;
        const formattedProductName = capitalizeFirstLetter(productName);


        // Obtém uma referência para a base de dados Firestore
        const db = getFirestore();

        // Cria uma referência para a coleção 'products'
        const productsCollectionRef = collection(db, 'products');

        // Consulta para encontrar o documento do produto pelo nome
        const q = query(productsCollectionRef, where('name', '==', formattedProductName));

        // Obtém os resultados da consulta
        const querySnapshot = await getDocs(q);

        // Verifica se encontrou algum documento
        if (!querySnapshot.empty) {
            // Exclui o documento do produto
            const productDocRef = querySnapshot.docs[0].ref;
            await deleteDoc(productDocRef);

            // Exclui as fotos associadas ao produto no armazenamento
            const productPhotos = querySnapshot.docs[0].data().photos;
            const storage = getStorage();
            const deletionPromises = [];
            productPhotos.forEach(photoUrl => {
                const photoRef = ref(storage, photoUrl);
                deletionPromises.push(deleteObject(photoRef));
            });

            // Espera todas as operações de exclusão serem concluídas
            await Promise.all(deletionPromises);

            return res.status(200).json({ success: true, message: 'Produto ' + productName + ' e suas fotos associadas foram removidos com sucesso.' });
        } else {
            return res.status(404).json({ success: false, message: 'Nenhum produto encontrado com o nome ' + productName + '.' });
        }
    } catch (error) {
        return res.status(500).json({ success: false, message: 'Erro ao remover o produto.', error: error.message });
    }
});
```

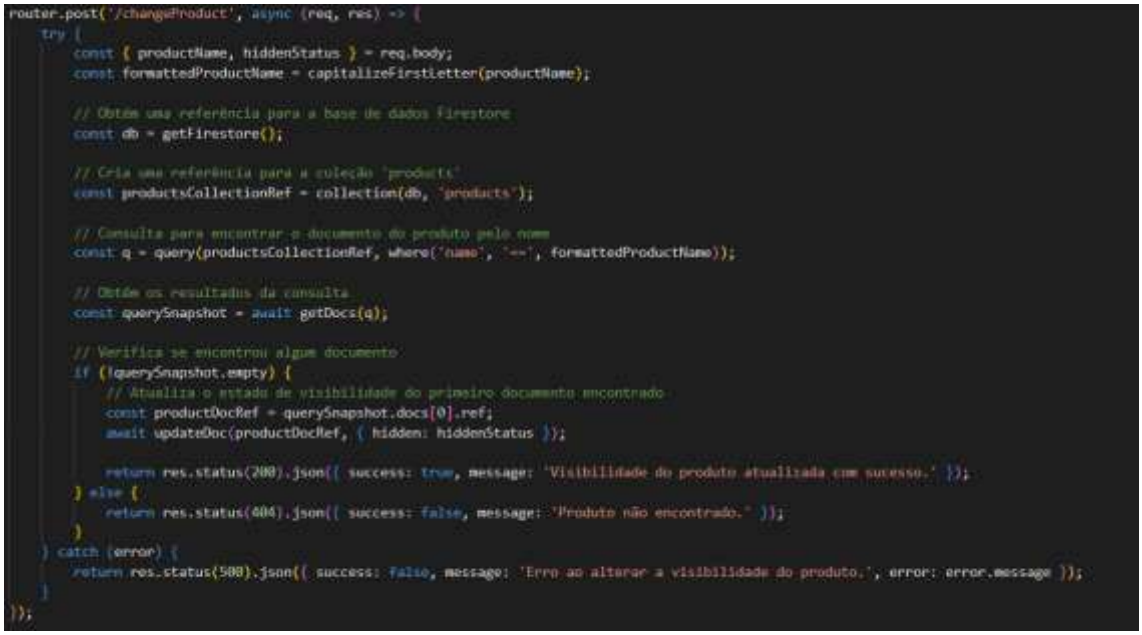
Hide/show products

Description: Allows you to hide the desired product.

How to access: On the admin page, select products, scroll to see the products and click on the eye icon to hide/show the desired product.

Technical details :

- **Route:** POST /changeProduct



```
router.post('/changeProduct', async (req, res) => {
    try {
        const { productName, hiddenStatus } = req.body;
        const formattedProductName = capitalizeFirstLetter(productName);

        // Obtém uma referência para a base de dados Firestore
        const db = getFirestore();

        // Cria uma referência para a coleção 'products'
        const productsCollectionRef = collection(db, 'products');

        // Consulta para encontrar o documento do produto pelo nome
        const q = query(productsCollectionRef, where('name', '==', formattedProductName));

        // Obtém os resultados da consulta
        const querySnapshot = await getDocs(q);

        // Verifica se encontrou algum documento
        if (!querySnapshot.empty) {
            // Atualiza o estado de visibilidade do primeiro documento encontrado
            const productDocRef = querySnapshot.docs[0].ref;
            await updateDoc(productDocRef, { hidden: hiddenStatus });

            return res.status(200).json({ success: true, message: 'Visibilidade do produto atualizada com sucesso.' });
        } else {
            return res.status(404).json({ success: false, message: 'Produto não encontrado.' });
        }
    } catch (error) {
        return res.status(500).json({ success: false, message: 'Erro ao alterar a visibilidade do produto.', error: error.message });
    }
});
```

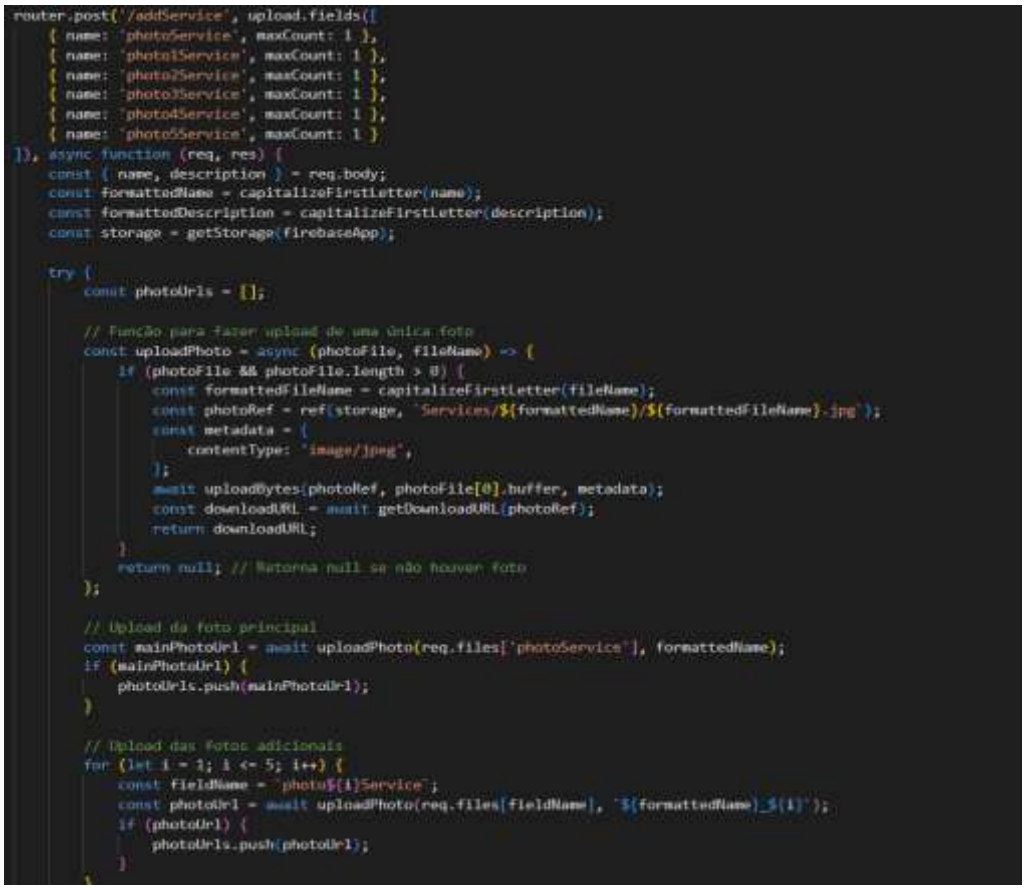
Add services

Description: Allows you to add services to the database.

How to access: On the admin page, select services, fill in the fields and click on the add service button **.**

Technical details:

- **Route:** POST /addService



```
router.post('/addService', upload.fields([
    { name: 'photoService', maxCount: 1 },
    { name: 'photo1Service', maxCount: 1 },
    { name: 'photo2Service', maxCount: 1 },
    { name: 'photo3Service', maxCount: 1 },
    { name: 'photo4Service', maxCount: 1 },
    { name: 'photo5Service', maxCount: 1 }
]), async function (req, res) {
    const { name, description } = req.body;
    const formattedName = capitalizeFirstLetter(name);
    const formattedDescription = capitalizeFirstLetter(description);
    const storage = getStorage(firebaseApp);

    try {
        const photoUrls = [];

        // Função para fazer upload de uma única foto
        const uploadPhoto = async (photoFile, fileName) => {
            if (photoFile && photoFile.length > 0) {
                const formattedFileName = capitalizeFirstLetter(fileName);
                const photoRef = ref(storage, `Services/${formattedName}/${formattedFileName}.jpg`);
                const metadata = {
                    contentType: 'image/jpeg',
                };
                await uploadBytes(photoRef, photoFile[0].buffer, metadata);
                const downloadURL = await getDownloadURL(photoRef);
                return downloadURL;
            }
            return null; // Retorna null se não houver foto
        };

        // Upload da foto principal
        const mainPhotoUrl = await uploadPhoto(req.files['photoService'], formattedName);
        if (mainPhotoUrl) {
            photoUrls.push(mainPhotoUrl);
        }

        // Upload das fotos adicionais
        for (let i = 1; i <= 5; i++) {
            const fieldName = `photo${i}Service`;
            const photoUrl = await uploadPhoto(req.files[fieldName], `${formattedName}_${i}`);
            if (photoUrl) {
                photoUrls.push(photoUrl);
            }
        }
```

```
        // Dados do serviço
        const sanitizedLowerCaseName = name.replace(/[^\s\wÀ-ÿ]/gi, '').toLowerCase();

        const serviceData = {
            name: formattedName,
            description: formattedDescription,
            photos: photoUrls,
            hidden: 'No',
            lowerCaseName: sanitizedLowerCaseName,
        };


        // Salvar os dados do serviço no Firestore
        const db = getFirestore(firebaseApp);
        const servicesCollection = collection(db, 'Services');
        await setDoc(doc(servicesCollection, formattedName), serviceData);

        return res.status(200).json({ success: true, message: 'Serviço adicionado com sucesso.' });
    } catch (error) {
        return res.status(500).json({ success: false, message: 'Erro ao adicionar Serviço.', error: error.message });
    }
});
```

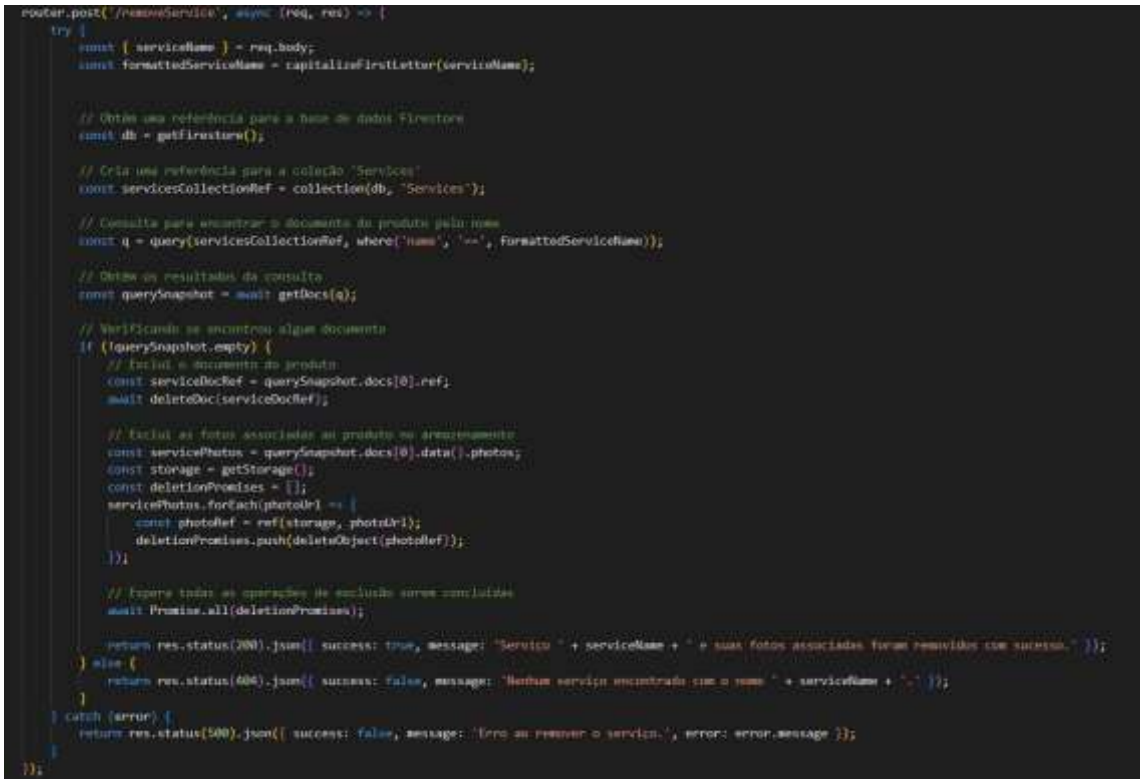
Remove services

Description: Allows you to delete services from the database.

How to access: On the admin page, select services, scroll to see the services and click on the trash icon to remove the desired service.

Technical details:

- **Route:** POST /removeService



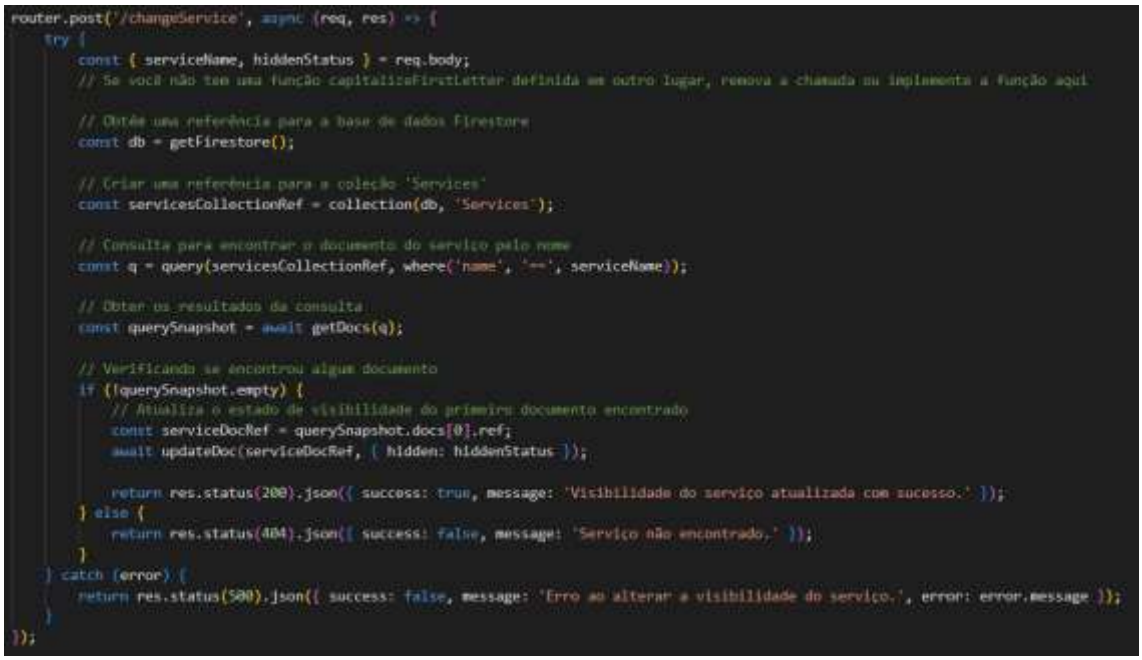
Hide/show services

Description: Allows you to hide the desired service.

How to access: On the admin page, in services, in the list of services, click on the hide/show service button.

Technical details:

- **Route:** POST /changeService



View orders older than 45 days

Description: Allows you to view only orders older than 45 days.

How to access: On the admin page, under orders, click the button see orders older than 45 days.

Technical details:

- **Route:** GET /showOrdersWithMoreThen45Days

```
router.get('/showOrdersWithMoreThan45Days', async function(req, res) {
    try {
        const db = getFirestore(firebaseApp);
        const ordersCollection = collection(db, 'orders');
        const snapshot = await getDocs(ordersCollection);

        const orders = [];
        const currentDate = new Date();
        const fortyFiveDaysAgo = new Date(currentDate.getTime() - (45 * 24 * 60 * 60 * 1000)); // Calcula a data 45 dias atrás

        snapshot.forEach(doc => {
            const orderData = doc.data();
            const orderDate = new Date(orderData.createdAt.toMillis());

            // Verifica se a encomenda foi feita há mais de 45 dias
            if (orderDate < fortyFiveDaysAgo) {
                orders.push(orderData);
            }
        });

        return res.status(200).json({ success: true, orders: orders });
    } catch (error) {
        return res.status(500).json({ success: false, message: 'Erro ao obter Encomendas.', error: error.message });
    }
});
```

Delete orders older than 45 days

Description: Allows you to delete orders that are more than 45 days old.

How to access: On the admin page, under orders, click the delete orders older than 45 days button.

Technical details:

- **Route:** POST /dropOrdersWithMoreThen45Days

```
router.post('/dropOrdersWithMoreThan45Days', async function(req, res) {
    try {
        const db = getFirestore(firebaseApp);
        const ordersCollection = collection(db, 'orders');
        const snapshot = await getDocs(ordersCollection);

        const currentDate = new Date();
        const fortyFiveDaysAgo = new Date(currentDate.getTime() - (45 * 24 * 60 * 60 * 1000)); // Calcula a data 45 dias atrás

        snapshot.forEach(async doc => {
            const orderData = doc.data();
            const orderDate = new Date(orderData.createdAt.toMillis());

            // Verifica se a encomenda foi feita há mais de 45 dias
            if (orderDate < fortyFiveDaysAgo) {
                // Exclui a encomenda
                await deleteDoc(doc.ref);
            }
        });

        return res.status(200).json({ success: true, message: 'Encomendas com mais de 45 dias excluídas com sucesso.' });
    } catch (error) {
        return res.status(500).json({ success: false, message: 'Erro ao excluir Encomendas.', error: error.message });
    }
});
```

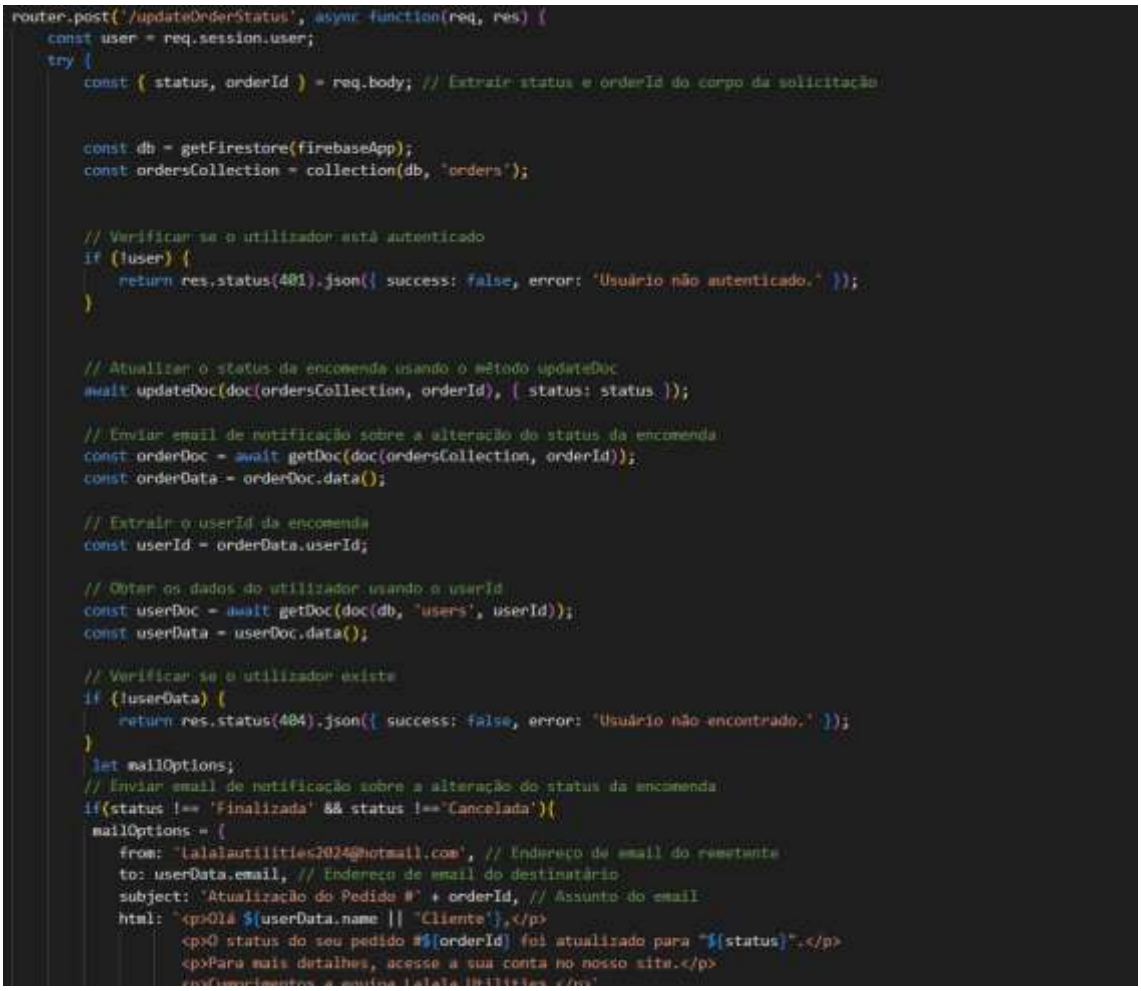
Modify order status

Description: Allows you to modify the status of orders.

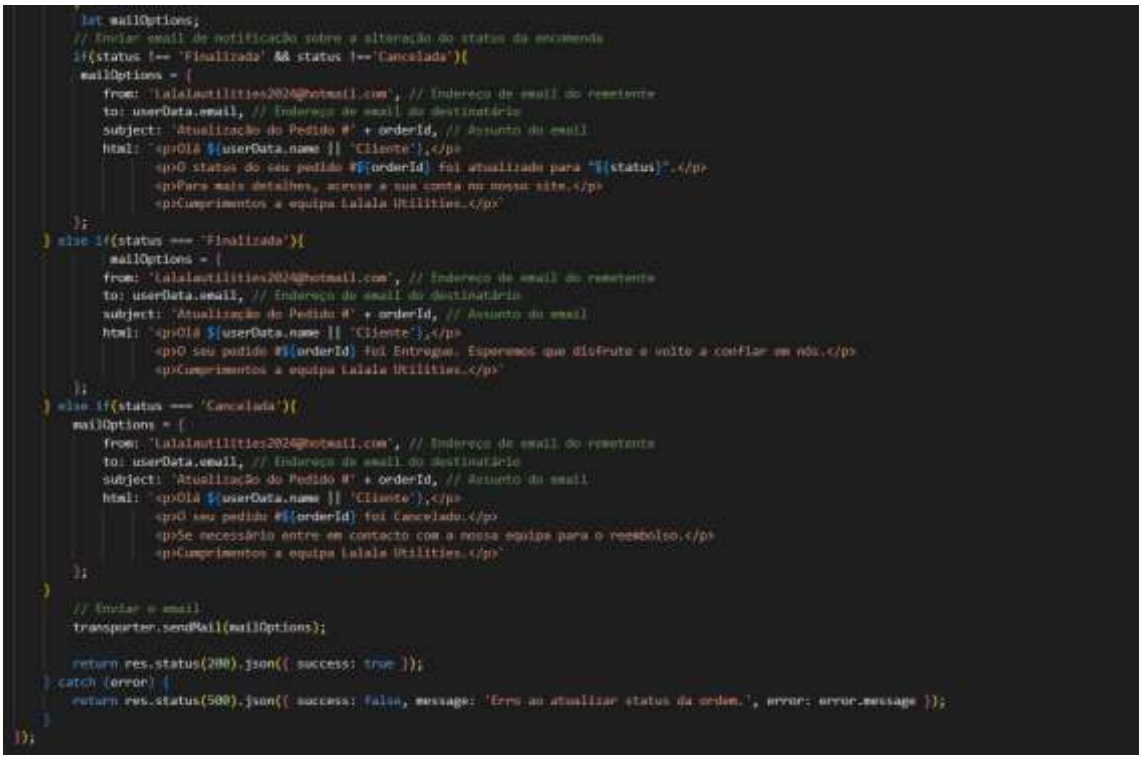
How to access: On the admin page, under orders, click on the order drop down menu and select the desired status.

Technical details:

- **Route:** POST /updateOrderStatus



```
router.post('/updateOrderStatus', async function(req, res) {
    const user = req.session.user;
    try {
        const { status, orderId } = req.body; // Extrair status e orderId do corpo da solicitação


        const db = getFirestore(firebaseApp);
        const ordersCollection = collection(db, 'orders');


        // Verificar se o utilizador está autenticado
        if (!user) {
            return res.status(401).json({ success: false, error: 'Usuário não autenticado.' });
        }


        // Atualizar o status da encomenda usando o método updateDoc
        await updateDoc(doc(ordersCollection, orderId), { status: status });

        // Enviar email de notificação sobre a alteração do status da encomenda
        const orderDoc = await getDoc(doc(ordersCollection, orderId));
        const orderData = orderDoc.data();

        // Extrair o userId da encomenda
        const userId = orderData.userId;

        // Obter os dados do utilizador usando o userId
        const userDoc = await getDoc(doc(db, 'users', userId));
        const userData = userDoc.data();

        // Verificar se o utilizador existe
        if (!userData) {
            return res.status(404).json({ success: false, error: 'Usuário não encontrado.' });
        }
         let mailOptions;
        // Enviar email de notificação sobre a alteração do status da encomenda
        if(status !== 'Finalizada' && status !=='Cancelada'){
         mailOptions = {
            from: 'lalalautilities2024@hotmail.com', // Endereço de email do remetente
            to: userData.email, // Endereço de email do destinatário
            subject: 'Atualização do Pedido #' + orderId, // Assunto do email
            html: `<p>Olá ${userData.name || 'Cliente'},</p>
                   <p>O status do seu pedido #${orderId} foi atualizado para "${status}".</p>
                   <p>Para mais detalhes, acesse a sua conta no nosso site.</p>
```



Search for orders by status

Description: Allows you to search for orders by their status.

How to access: On the admin page, in orders, click on the page drop down menu and select the desired status.

Technical details:

- **Route:** GET /searchOrdersByStatus

```
router.get('/searchOrdersByStatus', async function(req, res) {
    const status = req.query.status;
    const user = req.session.user;

    // Verificar se o utilizador está autenticado
    if (!user) {
        return res.status(401).json({ success: false, error: 'Usuário não autenticado.' });
    }

    try {
        const db = getFirestore();
        const ordersCollection = collection(db, 'orders');
        const snapshot = await getDocs(ordersCollection);

        const orders = [];
        snapshot.forEach(doc => {
            const orderData = doc.data();
            if (orderData.status === status) {
                orders.push(orderData);
            }
        });

        return res.status(200).json({ success: true, orders: orders });
    } catch (error) {
        return res.status(500).json({ success: false, message: 'Erro ao obter pedidos.', error: error.message });
    }
});
```

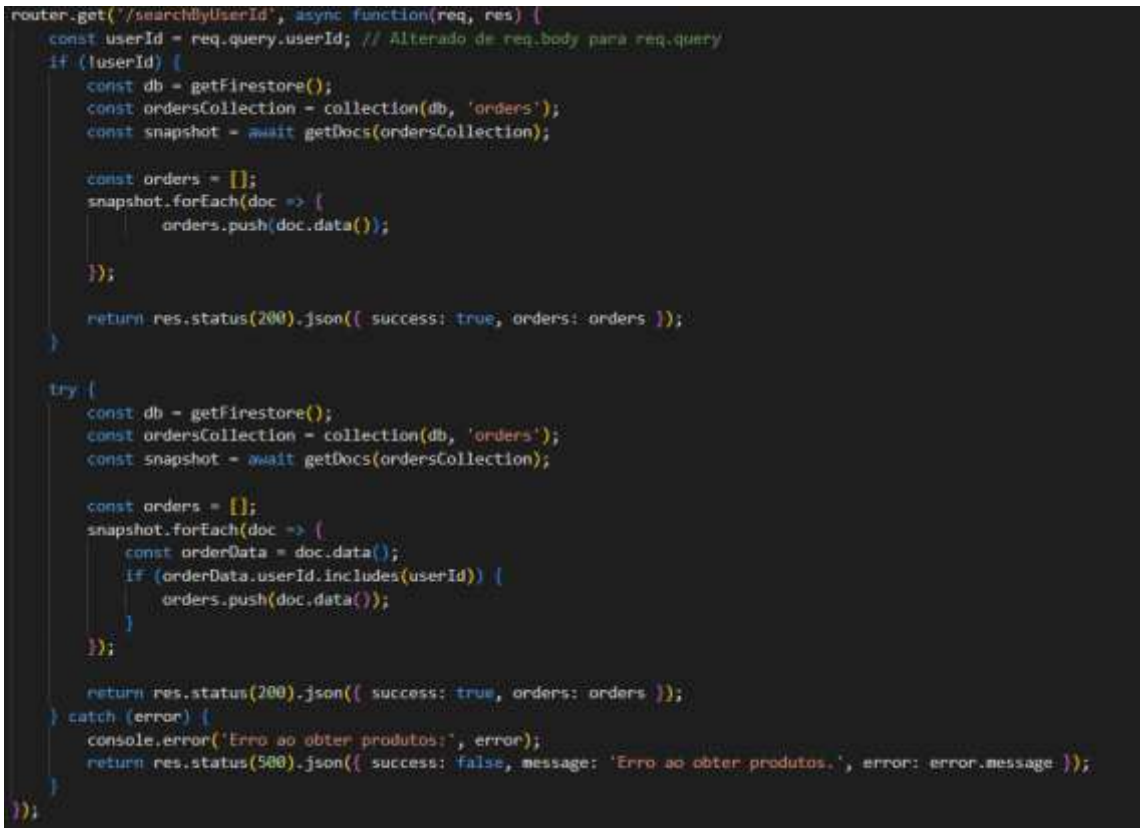
Search orders by user ID

Description: Allows you to search for orders by user ID.

How to access: On the admin page, in orders, in the search bar, enter the desired user ID.

Technical details:

- **Route:** GET /searchByUserId



```
router.get('/searchByUserId', async function(req, res) {
    const userId = req.query.userId; // Alterado de req.body para req.query
    if (!userId) {
        const db = getFirestore();
        const ordersCollection = collection(db, 'orders');
        const snapshot = await getDocs(ordersCollection);

        const orders = [];
        snapshot.forEach(doc => {
                orders.push(doc.data());

        });

        return res.status(200).json({ success: true, orders: orders });
    }

    try {
        const db = getFirestore();
        const ordersCollection = collection(db, 'orders');
        const snapshot = await getDocs(ordersCollection);

        const orders = [];
        snapshot.forEach(doc => {
            const orderData = doc.data();
            if (orderData.userId.includes(userId)) {
                orders.push(doc.data());
            }
        });

        return res.status(200).json({ success: true, orders: orders });
    } catch (error) {
        console.error('Erro ao obter produtos:', error);
        return res.status(500).json({ success: false, message: 'Erro ao obter produtos.', error: error.message });
    }
});
```

Conclusion

In this technical manual, we provide a detailed overview of the "Lalala Utilities" online store, developed with JavaScript, HTML, CSS, Bootstrap, Node.js, Express, jQuery and Ajax. The manual covered the following aspects:

- **Introduction** : Purpose of the manual and an overview of the application.
- **Description** : Overview of the online store, its main functionalities, technologies used, target audience and application architecture.
- **Prerequisites** : Tools and configurations required to configure and run the application.
- **Project Structure** : Organization of the project's main directories and files.
- **Installation and Configuration** : Steps to install and configure the application in the development environment.
- **Node.js and Express Server** : Server configuration and main routes.
- **Frontend Integration** : Use of jQuery and Ajax for asynchronous communication.
- **Main Features** : Detailed description of the online store's features.

Acknowledgements

A special thanks to my friends and colleagues who tested my project, gave their opinions and suggestions for improvement such as the application of product evaluation, among others.

For more information and additional resources, see the official documentation for the technologies used in the annexes.

Attachments

Webography

Multer: https://github.com/expressjs/multer

Firebase: https://firebase.google.com/docs?hl=en

Node.js: https://nodejs.org/docs/latest/api/

Express: https://devdocs.io/express/ **and** https://expressjs.com/

jQuery: https://api.jquery.com/

Crypto: https://nodejs.org/api/crypto.html

Axios: https://axios-http.com/docs/intro

Nodemailer: https://nodemailer.com/

Twitter Api: https://developer.x.com/en/docs/twitter-api

Google Api: https://developers.google.com/identity/protocols/oauth2?hl=pt-br