# Artisanal E-commerce

## Project overview & Description

pip install vitualenv

pip install flask flask_sqlalchemy

virtualenv venv

//////////////////////////////

create a folder "templates"

copy & paste index.html

create a folder static

copy & paste styles.css

in the static folder

create a folder structure by following

# Artisanal E-commerce

inside of static "images/product_images/"place the img files here""

```
.\venv\Scripts\activate
```

```
pip install Flask Flask-SQLAlchemy
```

```
python app.py
```

//////////////////////////////////////

in another cmd

```
.\venv\Scripts\activate
```

```
python init_db.py
```

//////////////////////////////////////

in another cmd

```
.\venv\Scripts\activate
```

# Artisanal E-commerce

python add_products_data.py

**Project Description:** Provide an overview of the artisanal e-commerce website, its purpose, and the unique selling points that set it apart from other e-commerce platforms.

- **User Types:** Identify and describe different user types, such as customers, artisans, and administrators.
- **User Registration:** Users should be able to create accounts with unique usernames and email addresses.
- **User Profiles:** Users should have the ability to create and manage their profiles, including personal information and preferences.
- **Authentication:** Implement secure user authentication and password recovery mechanisms.
- **Product Search:** Users should be able to search for products, filter by category, and sort by various criteria.
- **Product Details:** Provide detailed product listings, including images, descriptions, prices, and availability.
- **Shopping Cart:** Users should be able to add and remove items from their shopping cart, view the cart's contents, and proceed to checkout.
- **Checkout and Payment:** Implement a secure and straightforward checkout process with various payment options (e.g., credit card, PayPal).
- **Order History:** Users should have access to their order history and order status.
- **Reviews and Ratings:** Allow users to rate and review products.
- **Wishlist:** Users can create and manage a wishlist of products they're interested in.

- **Artisan Registration:** Artisans should be able to register their profiles, showcasing their products and craftsmanship.
- **Product Management:** Artisans can add, edit, and remove their products, including product details, images, and prices.
- **Order Management:** Artisans can manage orders for their products, including order processing and status updates.

# Artisanal E-commerce

## DATABASE DESIGN & CONNECTION

```
SQLite format 3
 @    `

                              ` .O}

Ý                        J Ý
k
!!�!tablecategoriescategoriesCREATE TABLE categories
             (categoryId INTEGER PRIMARY KEY,
             name TEXT
             )�:                        ,Wtablekartkart
CREATE TABLE kart
             (userId INTEGER,
             productId INTEGER,
             FOREIGN KEY(userId) REFERENCES users(userId),
             FOREIGN KEY(productId) REFERENCES
products(productId)

      )�xfCtableproductsproducts                          CREATE
TABLE products
             (productId INTEGER PRIMARY KEY,
```

# Artisanal E-commerce

```
        name TEXT,
        price REAL,
        description TEXT,
        image TEXT,
        stock INTEGER,
        categoryId INTEGER,
        FOREIGN KEY(categoryId) REFERENCES categories(categoryId)
        )⍰{ƒUtableusersusersCREATE TABLE users
        (userId INTEGER PRIMARY KEY,
        password TEXT,
        email TEXT,
        firstName TEXT,
        lastName TEXT,
        address1 TEXT,
        address2 TEXT,
        zipcode TEXT,
        city TEXT,
        state TEXT,
```

# Artisanal E-commerce

```
            country TEXT,
            phone TEXT
            )
```

```
import sqlite3


conn = sqlite3.connect('database.db')


conn.execute('''CREATE TABLE users
            (userId INTEGER PRIMARY KEY,
            password TEXT,
            email TEXT,
            firstName TEXT,
            lastName TEXT,
            address1 TEXT,
            address2 TEXT,
            zipcode TEXT,
            city TEXT,
            state TEXT,
            country TEXT,
            phone TEXT
            )''')

conn.execute('''CREATE TABLE products
            (productId INTEGER PRIMARY KEY,
            name TEXT,
            price REAL,
            description TEXT,
            image TEXT,
            stock INTEGER,
            categoryId INTEGER,
            FOREIGN KEY(categoryId) REFERENCES
categories(categoryId)
            )''')

conn.execute('''CREATE TABLE kart
            (userId INTEGER,
            productId INTEGER,
            FOREIGN KEY(userId) REFERENCES users(userId),
            FOREIGN KEY(productId) REFERENCES
products(productId)
            )''')

conn.execute('''CREATE TABLE categories
            (categoryId INTEGER PRIMARY KEY,
            name TEXT
            )''')



conn.close()
```

## FRONT & BACK END DEVELOPMENT

# Artisanal E-commerce

## CSS

```
.Title{
        font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode', Geneva, Verdana, sans-serif;
        font-size: 1.6rem;
}
#itemImage {
        height: 200px;
        width: 150px;
}

.display {
        margin-top: 20px;
        margin-left: 20px;
        margin-right: 20px;
        margin-bottom: 20px;
}
```

# Artisanal E-commerce

```
table {
        border-spacing: 20px;
}

#productName {
        text-align: center;
        font-weight: bold;
}

#productPrice {
        text-align: center;
}

.displayCategory ul li {
        font-size: 20px;
}
```

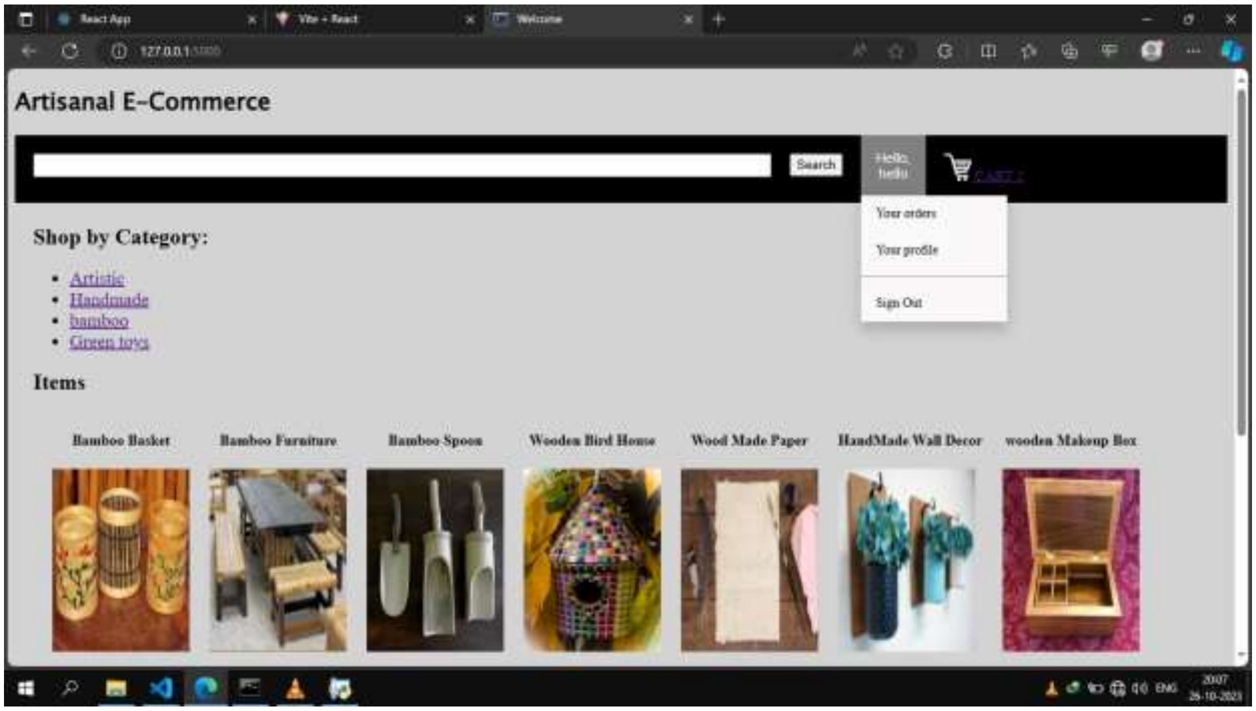
# Artisanal E-commerce

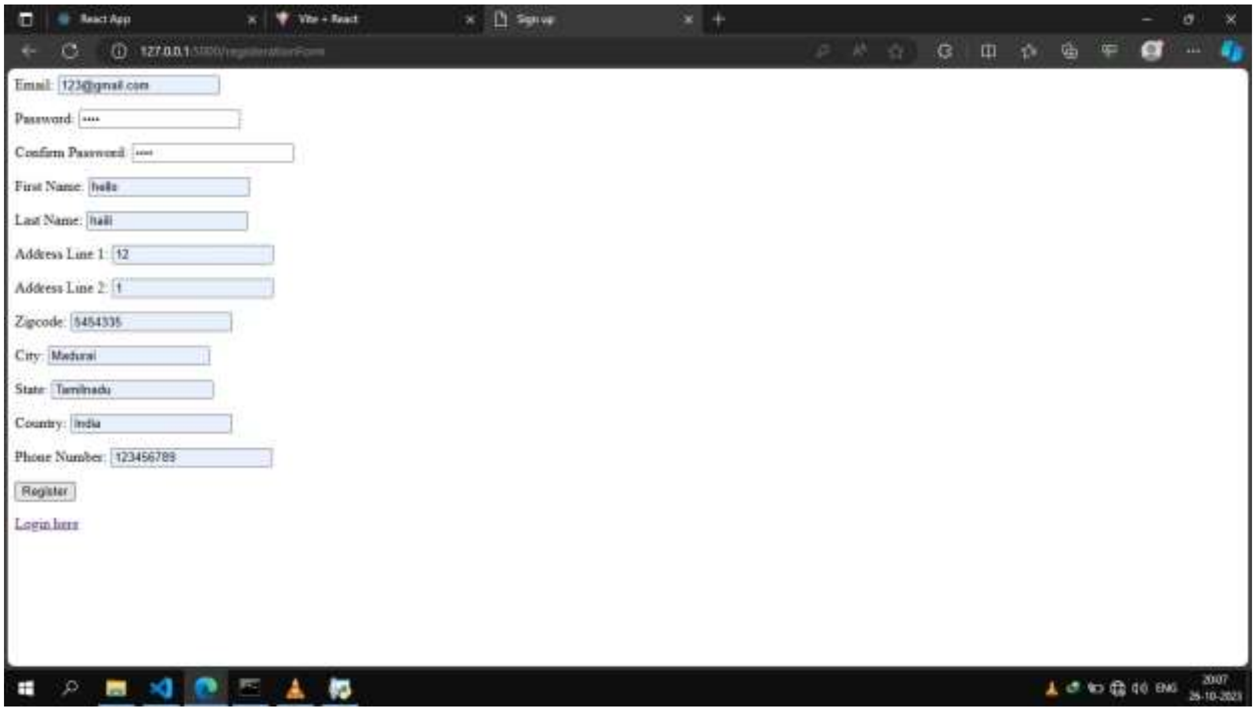
```
body{
        background-color: lightgray;
}
#title{
        background-color: cornsilk;
}
```

## JAVA SCRIPT

```
        function validate() {
   var pass = 
document.getElementById("password").value;
   var cpass = 
document.getElementById("cpassword").value;
   if (pass == cpass) {
      return true;
   } else {
      alert("Passwords do not match");
      return false;
   }
}
function validate() {
    var pass =
document.getElementById("newpassword").value;
    var cpass =
document.getElementById("cpassword").value;
    if (pass == cpass) {
        return true;
    } else {
        alert("Passwords do not match!");
        return false;
    }
}
```

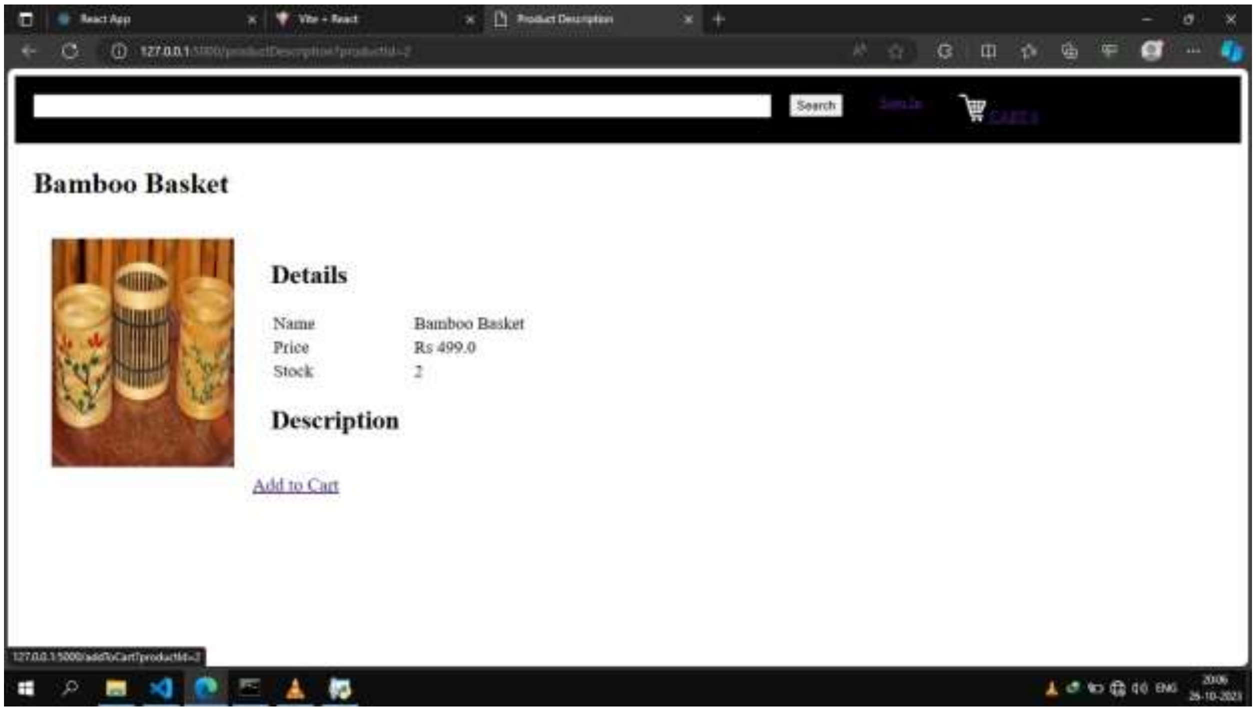
# Artisanal E-commerce





|  |  |
|---|---|
| Email: | 123@gmail.com |
| Password: | **** |
| Confirm Password: | **** |
| First Name: | hello |
| Last Name: | halli |
| Address Line 1: | 12 |
| Address Line 2: | 1 |
| Zipcode: | 5454335 |
| City: | Madurai |
| State: | Tamilnadu |
| Country: | India |
| Phone Number: | 123456789 |

# Artisanal E-commerce



Bamboo Basket

Details

| | |
|---|---|
| Name | Bamboo Basket |
| Price | Rs 499.0 |
| Stock | 2 |

Description

Add to Cart

## PYTHON SOURCES FILES

```
from flask import *

import sqlite3, hashlib, os

from werkzeug.utils import secure_filename

from instamojo_wrapper import Instamojo

import requests


app = Flask(__name__)
```

# Artisanal E-commerce

```
app.secret_key = 'random string'
UPLOAD_FOLDER = 'static/uploads'
ALLOWED_EXTENSIONS = set(['jpeg', 'jpg', 'png', 'gif'])
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER


#Home page
@app.route("/")
def root():
    loggedIn, firstName, noOfItems = getLoginDetails()
    with sqlite3.connect('database.db') as conn:
        cur = conn.cursor()
        cur.execute('SELECT productId, name, price, 
description, image, stock FROM products')
        itemData = cur.fetchall()
        cur.execute('SELECT categoryId, name FROM 
categories')
        categoryData = cur.fetchall()
    itemData = parse(itemData)
    return render_template('home.html',
itemData=itemData, loggedIn=loggedIn,
firstName=firstName, noOfItems=noOfItems,
categoryData=categoryData)


#Fetch user details if logged in
def getLoginDetails():
    with sqlite3.connect('database.db') as conn:
        cur = conn.cursor()
        if 'email' not in session:
            loggedIn = False
            firstName = ''
            noOfItems = 0
        else:
            loggedIn = True
            cur.execute("SELECT userId, firstName FROM users
WHERE email = '" + session['email'] + "'")
            userId, firstName = cur.fetchone()
        cur.execute("SELECT count(productId) FROM kart
WHERE userId = " + str(userId))
        noOfItems = cur.fetchone()[0]
  conn.close()
  return (loggedIn, firstName, noOfItems)

#Add item to cart
@app.route("/addItem", methods=["GET", "POST"])
def addItem():
  if request.method == "POST":
    name = request.form['name']
    price = float(request.form['price'])
    description = request.form['description']
    stock = int(request.form['stock'])
    categoryId = int(request.form['category'])

    #Upload image
    image = request.files['image']
```

# Artisanal E-commerce

```
        if image and allowed_file(image.filename):
            filename = secure_filename(image.filename)

image.save(os.path.join(app.config['UPLOAD_FOLDER'],
filename))
        imagename = filename
        with sqlite3.connect('database.db') as conn:
            try:
                cur = conn.cursor()
                cur.execute('''INSERT INTO products (name, price,
description, image, stock, categoryId) VALUES (?, ?, ?, ?, ?,
?)''', (name, price, description, imagename, stock,
categoryId))
                conn.commit()
                msg="Added successfully"
            except:
                msg="Error occured"
                conn.rollback()
        conn.close()
        print(msg)

        return redirect(url_for('root'))


#Remove item from cart

@app.route("/removeItem")

def removeItem():

    productId = request.args.get('productId')

    with sqlite3.connect('database.db') as conn:

        try:

            cur = conn.cursor()

            cur.execute('DELETE FROM products WHERE 
productID = ' + productId)

            conn.commit()

            msg = "Deleted successsfully"

        except:

            conn.rollback()

            msg = "Error occured"

    conn.close()
```

```
 print(msg)

 return redirect(url_for('root'))


#Display all items of a category

@app.route("/displayCategory")

def displayCategory():

    loggedIn, firstName, noOfItems = getLoginDetails()

    categoryId = request.args.get("categoryId")

    with sqlite3.connect('database.db') as conn:

        cur = conn.cursor()

        cur.execute("SELECT products.productId,
products.name, products.price, products.image,
categories.name FROM products, categories WHERE
products.categoryId = categories.categoryId AND
categories.categoryId = " + categoryId)

        data = cur.fetchall()

    conn.close()

    categoryName = data[0][4]
```

```
        data = parse(data)
        return render_template('displayCategory.html', data=data, loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems, categoryName=categoryName)


@app.route("/account/profile")
def profileHome():
    if 'email' not in session:
        return redirect(url_for('root'))
    loggedIn, firstName, noOfItems = getLoginDetails()
    return render_template("profileHome.html", loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems)


@app.route("/account/profile/edit")
def editProfile():
    if 'email' not in session:
        return redirect(url_for('root'))
```

# Artisanal E-commerce

```
    loggedIn, firstName, noOfItems = getLoginDetails()
    with sqlite3.connect('database.db') as conn:
        cur = conn.cursor()
        cur.execute("SELECT userId, email, firstName,
lastName, address1, address2, zipcode, city, state,
country, phone FROM users WHERE email = '" +
session['email'] + "'")
        profileData = cur.fetchone()
    conn.close()
    return render_template("editProfile.html",
profileData=profileData, loggedIn=loggedIn,
firstName=firstName, noOfItems=noOfItems)


@app.route("/account/profile/changePassword",
methods=["GET", "POST"])
def changePassword():
    if 'email' not in session:
        return redirect(url_for('loginForm'))
    if request.method == "POST":
```

# Artisanal E-commerce

```
        oldPassword = request.form['oldpassword']
        oldPassword =
hashlib.md5(oldPassword.encode()).hexdigest()
        newPassword = request.form['newpassword']
        newPassword =
hashlib.md5(newPassword.encode()).hexdigest()
        with sqlite3.connect('database.db') as conn:
            cur = conn.cursor()
            cur.execute("SELECT userId, password FROM users
WHERE email = '" + session['email'] + "'")
            userId, password = cur.fetchone()
            if (password == oldPassword):
                try:
                    cur.execute("UPDATE users SET password = ?
WHERE userId = ?", (newPassword, userId))
                    conn.commit()
                    msg="Changed successfully"
                except:
                    conn.rollback()
                    msg = "Failed"
                return render_template("changePassword.html",
msg=msg)
            else:
                msg = "Wrong password"
        conn.close()
        return render_template("changePassword.html",
msg=msg)
    else:
        return render_template("changePassword.html")


@app.route("/updateProfile", methods=["GET", "POST"])
def updateProfile():
    if request.method == 'POST':
        email = request.form['email']
        firstName = request.form['firstName']
        lastName = request.form['lastName']
```

# Artisanal E-commerce

```
    address1 = request.form['address1']
    address2 = request.form['address2']
    zipcode = request.form['zipcode']
    city = request.form['city']
    state = request.form['state']
    country = request.form['country']
    phone = request.form['phone']
    with sqlite3.connect('database.db') as con:
        try:
            cur = con.cursor()
            cur.execute('UPDATE users SET firstName = ?, 
lastName = ?, address1 = ?, address2 = ?, zipcode = ?, city 
= ?, state = ?, country = ?, phone = ? WHERE email = ?', 
(firstName, lastName, address1, address2, zipcode, city, 
state, country, phone, email))


            con.commit()
            msg = "Saved Successfully"
```

```
                except:
                    con.rollback()
                    msg = "Error occured"
        con.close()
        return redirect(url_for('editProfile'))

@app.route("/loginForm")
def loginForm():
    if 'email' in session:
        return redirect(url_for('root'))
    else:
        return render_template('login.html', error='')

@app.route("/login", methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        if is_valid(email, password):
            session['email'] = email
            return redirect(url_for('root'))
        else:
            error = 'Invalid UserId / Password'
            return render_template('login.html', error=error)

@app.route("/productDescription")
def productDescription():
    loggedIn, firstName, noOfItems = getLoginDetails()
    productId = request.args.get('productId')
    with sqlite3.connect('database.db') as conn:
        cur = conn.cursor()
        cur.execute('SELECT productId, name, price,
description, image, stock FROM products WHERE
productId = ' + productId)
        productData = cur.fetchone()
    conn.close()
    return render_template("productDescription.html",
data=productData, loggedIn = loggedIn, firstName =
firstName, noOfItems = noOfItems)


@app.route("/addToCart")
def addToCart():
    if 'email' not in session:
        return redirect(url_for('loginForm'))
    else:
        productId = int(request.args.get('productId'))
        with sqlite3.connect('database.db') as conn:
            cur = conn.cursor()
            cur.execute("SELECT userId FROM users WHERE
email = '" + session['email'] + "'")
            userId = cur.fetchone()[0]
            try:
```

```
            cur.execute("INSERT INTO kart (userId, productId)
VALUES (?, ?)", (userId, productId))
            conn.commit()
            msg = "Added successfully"
         except:
            conn.rollback()
            msg = "Error occured"
      conn.close()
      return redirect(url_for('root'))

@app.route("/cart")
def cart():
   if 'email' not in session:
      return redirect(url_for('loginForm'))
   loggedIn, firstName, noOfItems = getLoginDetails()
   email = session['email']
   with sqlite3.connect('database.db') as conn:
      cur = conn.cursor()
```

```
        cur.execute("SELECT userId FROM users WHERE email
= '" + email + "'")

        userId = cur.fetchone()[0]

        cur.execute("SELECT products.productId,
products.name, products.price, products.image FROM
products, kart WHERE products.productId = kart.productId
AND kart.userId = " + str(userId))

        products = cur.fetchall()

    totalPrice = 0

    for row in products:

        totalPrice += row[2]

    return render_template("cart.html", products =
products, totalPrice=totalPrice, loggedIn=loggedIn,
firstName=firstName, noOfItems=noOfItems)


@app.route("/checkout")

def checkout():

    if 'email' not in session:

        return redirect(url_for('loginForm'))
    loggedIn, firstName, noOfItems = getLoginDetails()
    email = session['email']
    with sqlite3.connect('database.db') as conn:
        cur = conn.cursor()
        cur.execute("SELECT userId FROM users WHERE email
= '" + email + "'")
        userId = cur.fetchone()[0]
        cur.execute("SELECT products.productId,
products.name, products.price, products.image FROM
products, kart WHERE products.productId = kart.productId
AND kart.userId = " + str(userId))
        products = cur.fetchall()
    totalPrice = 0
    for row in products:
        totalPrice += row[2]
    return render_template("checkout.html", products =
products, totalPrice=totalPrice, loggedIn=loggedIn,
firstName=firstName, noOfItems=noOfItems)
```

```
@app.route("/instamojo")
def instamojo():
    return render_template("instamojo.html")


@app.route("/removeFromCart")
def removeFromCart():
    if 'email' not in session:
        return redirect(url_for('loginForm'))
    email = session['email']
    productId = int(request.args.get('productId'))
    with sqlite3.connect('database.db') as conn:
        cur = conn.cursor()
        cur.execute("SELECT userId FROM users WHERE email
= '" + email + "'")
        userId = cur.fetchone()[0]
        try:
            cur.execute("DELETE FROM kart WHERE userId = " +
str(userId) + " AND productId = " + str(productId))
```

```
            conn.commit()

            msg = "removed successfully"

        except:

            conn.rollback()

            msg = "error occured"

    conn.close()

    return redirect(url_for('root'))


@app.route("/logout")

def logout():

    session.pop('email', None)

    return redirect(url_for('root'))


def is_valid(email, password):

    con = sqlite3.connect('database.db')

    cur = con.cursor()

    cur.execute('SELECT email, password FROM users')
    data = cur.fetchall()
    for row in data:
        if row[0] == email and row[1] == hashlib.md5(password.encode()).hexdigest():
            return True
    return False


@app.route("/register", methods = ['GET', 'POST'])
def register():
    if request.method == 'POST':
        #Parse form data
        password = request.form['password']
        email = request.form['email']
        firstName = request.form['firstName']
        lastName = request.form['lastName']
        address1 = request.form['address1']
        address2 = request.form['address2']
        zipcode = request.form['zipcode']
        city = request.form['city']

        state = request.form['state']

        country = request.form['country']

        phone = request.form['phone']


        with sqlite3.connect('database.db') as con:

            try:

                cur = con.cursor()

                cur.execute('INSERT INTO users (password, email,
firstName, lastName, address1, address2, zipcode, city,
state, country, phone) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)',
(hashlib.md5(password.encode()).hexdigest(), email,
firstName, lastName, address1, address2, zipcode, city,
state, country, phone))


                con.commit()


                msg = "Registered Successfully"

            except:
                con.rollback()

                msg = "Error occured"

        con.close()

        return render_template("login.html", error=msg)


@app.route("/registerationForm")

def registrationForm():

    return render_template("register.html")


def allowed_file(filename):

    return '.' in filename and \

            filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS


def parse(data):

    ans = []

    i = 0

    while i < len(data):
```

# Artisanal E-commerce

```
        curr = []
        for j in range(7):
            if i >= len(data):
                break
            curr.append(data[i])
            i += 1
        ans.append(curr)
    return ans


if __name__ == '__main__':
    app.run(debug=True)
```

## project overview

The artisanal e-commerce website is a platform designed to connect artisans and craft enthusiasts. It provides a marketplace for artisans to showcase their unique, handcrafted products and allows customers to discover and purchase these one-of-a-kind items. The website fosters a community of creators and buyers who appreciate craftsmanship, creativity, and quality.

**Key Features:**

1. **User-Friendly Shopping Experience:**
    - User registration and profiles.
    - Product search, filtering, and sorting.

# Artisanal E-commerce

- Detailed product listings with images, descriptions, and pricing.
- Shopping cart for easy item management.
- Secure and streamlined checkout process.

2. **Artisan Empowerment:**
   - Artisan registration and profile management.
   - Product management with product details, images, and prices.
   - Order management for artisans, including processing and status updates.
3. **Administrator Control:**
   - User account management, including roles and permissions.
   - Comprehensive product management, including addition, editing, and removal.
   - Centralized order management with status updates.
   - Content management for website updates.
   - Reporting and analytics for performance tracking.
4. **Product Management:**
   - Organized product categories and attributes.
   - Inventory management to track product availability.
   - User-friendly image upload for high-quality product images.
5. **Security and Privacy:**
   - Data encryption for user information and payments.
   - Compliance with privacy regulations and clear privacy policies.
6. **Performance and Scalability:**
   - Fast loading times for desktop and mobile users.
   - Scalability for future growth.
7. **Mobile Responsiveness:**
   - A fully functional and responsive design for mobile devices.
8. **Payment Processing:**
   - Integration with secure payment gateways.
9. **SEO and Marketing:**
   - Search engine optimization for better visibility.
   - Marketing features like promotional banners and discount codes.
10. **Legal and Compliance:**
    - Adherence to e-commerce laws and regulations.
    - Display of terms and conditions, return policies, and other legal information.
11. **Reporting and Analytics:**
    - Analytics tools for monitoring website performance and user behavior.

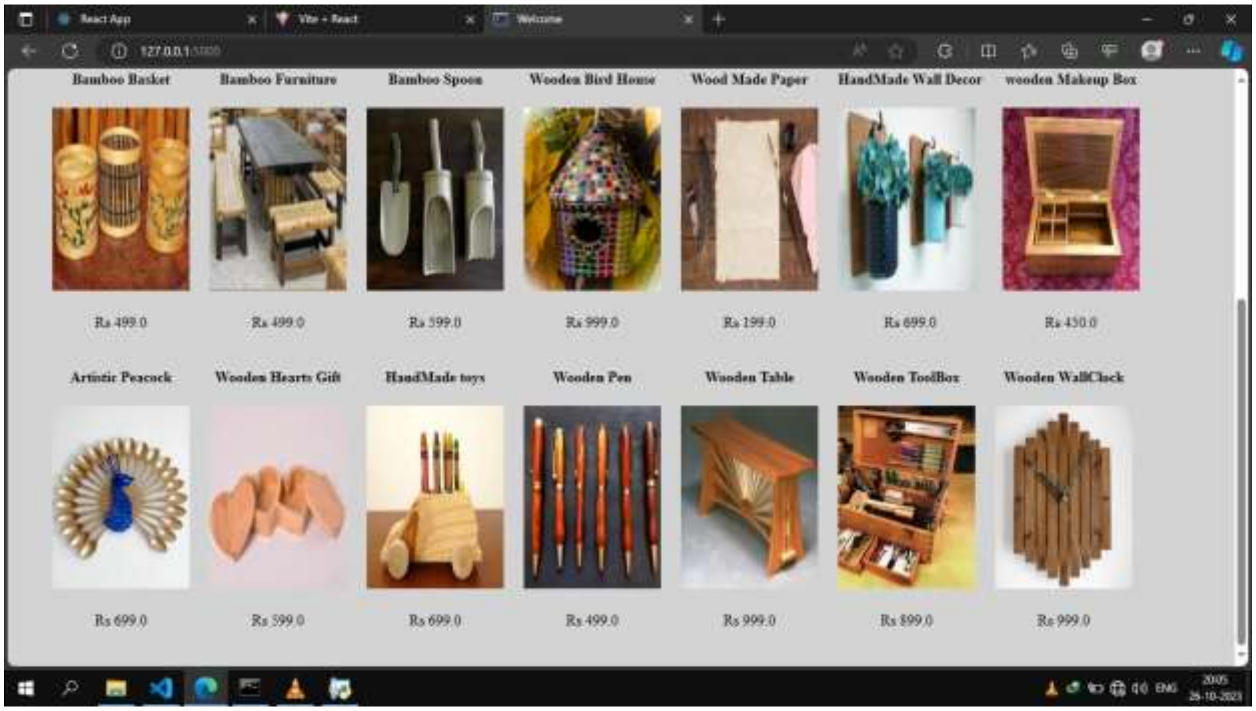
**Budget and Timeline:** A defined project budget and timeline have been established, covering development, testing, and the website's launch.

# Artisanal E-commerce

**Support and Maintenance:** Post-launch, the project includes plans for ongoing support and maintenance to ensure a smooth and reliable user experience.

The artisanal e-commerce website aims to foster a vibrant community of artisans and art lovers, providing a marketplace for their creations while emphasizing security, user-friendliness, and quality. It aspires to be a hub where creativity, craftsmanship, and commerce seamlessly converge.

## SAMPLE OUTPUT


React App
Vite + React
Welcome
127.0.0.1
Bamboo Basket
Rs 499.0
Bamboo Furniture
Rs 499.0
Bamboo Spoon
Rs 599.0
Wooden Bird House
Rs 999.0
Wood Made Paper
Rs 199.0
HandMade Wall Decor
Rs 699.0
wooden Makeup Box
Rs 450.0
Artistic Peacock
Rs 699.0
Wooden Hearts Gift
Rs 599.0
HandMade toys
Rs 699.0
Wooden Pen
Rs 499.0
Wooden Table
Rs 999.0
Wooden ToolBox
Rs 899.0
Wooden WallClock
Rs 999.0
ENG
20:05
26-10-2023