1. **Personalized Recommendations (Machine Learning Integration)**

   Code Snippet: Collaborative Filtering Algorithm

```python
# Python snippet for generating personalized product recommendations using collaborative filtering
from sklearn.neighbors import NearestNeighbors
import pandas as pd

# Sample user-item interaction data
data = {
    'user_id': [1, 1, 2, 2, 3, 3, 4, 4],
    'product_id': [101, 102, 101, 103, 102, 104, 103, 104],
    'rating': [5, 4, 3, 5, 4, 2, 5, 3]
}

df = pd.DataFrame(data)

# Pivot table for user-item matrix
user_item_matrix = df.pivot(index='user_id', columns='product_id', values='rating').fillna(0)

# KNN model for finding similar users
model_knn = NearestNeighbors(metric='cosine', algorithm='brute')
model_knn.fit(user_item_matrix)

# Function to get recommendations for a user
def get_recommendations(user_id, n_recommendations=5):
    distances, indices = model_knn.kneighbors(user_item_matrix.iloc[user_id-1, :].values.reshape(1, -1), n_neighbors=n_recommendations+1)
    similar_users = indices.flatten()[1:]
    recommended_products = user_item_matrix.iloc[similar_users].mean(axis=0).sort_values(ascending=False).index.tolist()
    return recommended_products[:n_recommendations]

# Example: Get recommendations for user_id = 1
print(get_recommendations(1))
```

Explanation:

- This snippet demonstrates how collaborative filtering was integrated to offer personalized product recommendations.
- The algorithm finds users with similar preferences and recommends products based on their behavior.
- Integrated with the backend using Python and released as a microservice.

2. **Smart Search (Algolia Integration)**

Code Snippet: Frontend Search Component

```jsx
// React component for smart search using Algolia
import React, { useState } from 'react';
import algoliasearch from 'algoliasearch/lite';
import { InstantSearch, SearchBox, Hits } from 'react-instantsearch-dom';

// Initialize Algolia client
const searchClient = algoliasearch('YourAppID', 'YourSearchOnlyAPIKey');

const SmartSearch = () => {
  const [query, setQuery] = useState('');

  return (
    <InstantSearch searchClient={searchClient} indexName="products">
      <SearchBox
        translations={{ placeholder: 'Search for products...' }}
        onChange={(e) => setQuery(e.target.value)}
      />
      <Hits hitComponent={ProductHit} />
    </InstantSearch>
  );
};

// Component to display search results
const ProductHit = ({ hit }) => (
  <div className="search-result">
    <img src={hit.image} alt={hit.name} />
    <h3>{hit.name}</h3>
    <p>{hit.description}</p>
    <p>${hit.price}</p>
  </div>
);

export default SmartSearch;
```

Explanation:

- This React component integrates Algolia's search-as-you-type functionality.
- It provides fast, typo-tolerant search results with a seamless user experience.
- The Hits component displays product details, including images, names, descriptions, and prices.

3. **Automated Checkout (Stripe Integration)**
   Code Snippet: Backend Payment Processing

```
// Node.js snippet for processing payments using Stripe
const stripe = require('stripe')('YourStripeSecretKey');
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

app.use(bodyParser.json());

// Endpoint to create a payment intent
app.post('/create-payment-intent', async (req, res) => {
  const { amount, currency, payment_method } = req.body;

  try {
    const paymentIntent = await stripe.paymentIntents.create({
      amount,
      currency,
      payment_method,
      confirm: true,
    });

    res.status(200).json({ success: true, paymentIntent });
  } catch (error) {
    res.status(400).json({ success: false, error: error.message });
  }
});

// Start server
app.listen(3000, () => {
  console.log('Server running on http://localhost:3000');
});

```

Explanation:

- This backend API endpoint handles payment processing using Stripe.
- It creates a PaymentIntent to securely process the transaction.
- Integrated with the frontend to allow one-click checkout for users.

4. **Responsive Design (React and CSS)**
   Code Snippet: Responsive Product Card Component

```jsx
// React component for a responsive product card
import React from 'react';
import './ProductCard.css';

const ProductCard = ({ product }) => {
  return (
    <div className="product-card">
      <img src={product.image} alt={product.name} className="product-image" />
      <h3 className="product-name">{product.name}</h3>
      <p className="product-description">{product.description}</p>
      <p className="product-price">${product.price}</p>
      <button className="add-to-cart-btn">Add to Cart</button>
    </div>
  );
};

export default ProductCard;
```

CSS for Responsive Design

```css
/* ProductCard.css */
.product-card {
  border: 1px solid #ddd;
  border-radius: 8px;
  padding: 16px;
  text-align: center;
  width: 100%;
  max-width: 300px;
  margin: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.product-image {
  width: 100%;
  height: auto;
  border-radius: 8px;
}

.product-name {
  font-size: 1.2rem;
  margin: 10px 0;
}

.product-description {
  font-size: 0.9rem;
  color: #555;
}

.product-price {
  font-size: 1.1rem;
  font-weight: bold;
  color: #333;
}

.add-to-cart-btn {
  background-color: #007bff;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 5px;
  cursor: pointer;
  margin-top: 10px;
}

.add-to-cart-btn:hover {
  background-color: #0056b3;
}

/* Responsive design */
@media (max-width: 768px) {
  .product-card {
    max-width: 100%;
  }
}
```

Explanation:

- This React code and CSS show how responsive design was implemented for product cards.
- The layout scales gracefully with different screen sizes, delivering the same user experience regardless of devices.

5. Testing (Cypress End-to-End Test)
   Code Snippet: End-to-End Test for Checkout Flow

```
// Cypress test for the checkout process
describe('Checkout Flow', () => {
  it('Successfully completes a purchase', () => {
    cy.visit('/');
    cy.get('.product-card').first().click();
    cy.get('.add-to-cart-btn').click();
    cy.get('.cart-icon').click();
    cy.get('.checkout-btn').click();
    cy.get('#payment-form').within(() => {
      cy.get('#card-number').type('4242424242424242');
      cy.get('#expiry-date').type('12/25');
      cy.get('#cvc').type('123');
    });
    cy.get('.submit-payment-btn').click();
    cy.contains('Thank you for your purchase!').should('be.visible');
  });
});
```

Explanation:

- This Cypress test automates the checkout process, ensuring that the flow works as expected.
- It simulates the interaction of a real user, from adding an item to the shopping cart, through entering payment details, up to completing the purchase.