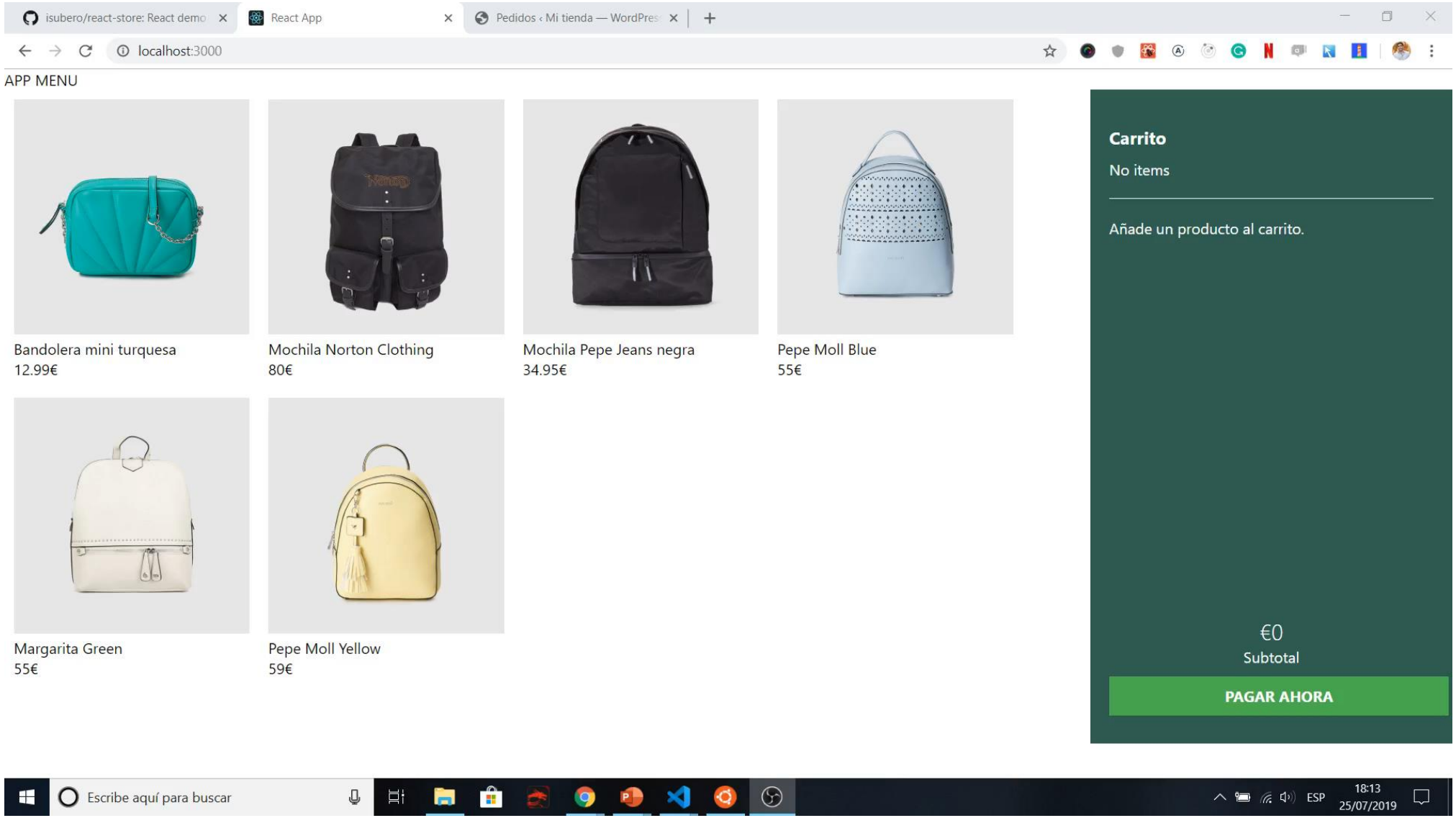
Crea una SPA con WooCommerce y React JS

Isaías Subero

Ventajas de una SPA

- Rápida y responsiva.
- Mejor experiencia de usuario.
- Evitamos peticiones innecesarias de assets.
- Mejor aprovechamiento de los recursos.
- Ahorro en costos de servidor.
- Potencialidades de PWA.
- El usuario no necesita instalar una app.

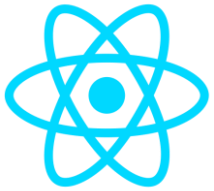
¿Qué estaremos construyendo?



¿Qué estaremos construyendo?

- Vista de productos.
- Vista de detalle de productos.
- Agregar productos al carrito.
- Eliminar productos del carrito.
- Recalcular automáticamente subtotal del carrito.
- Comunicarnos con la API de WooCommerce para acceder a la data de los productos y crear pedidos (orders).
- Aprendiendo rutas.
- React Context para manejar el estado global de la aplicación.
- CSS Modules.
- ¡Sin recargar la página! ☺

¿Qué estaremos usando?



React
Librería Javascript para crear interfaces de usuario.

WooCommerce
Plugin de WordPress para crear tiendas online.

React Router
Colección de componentes de navegación.

CRA

Create React App
Utilidad para crear un ambiente de desarrollo de React.

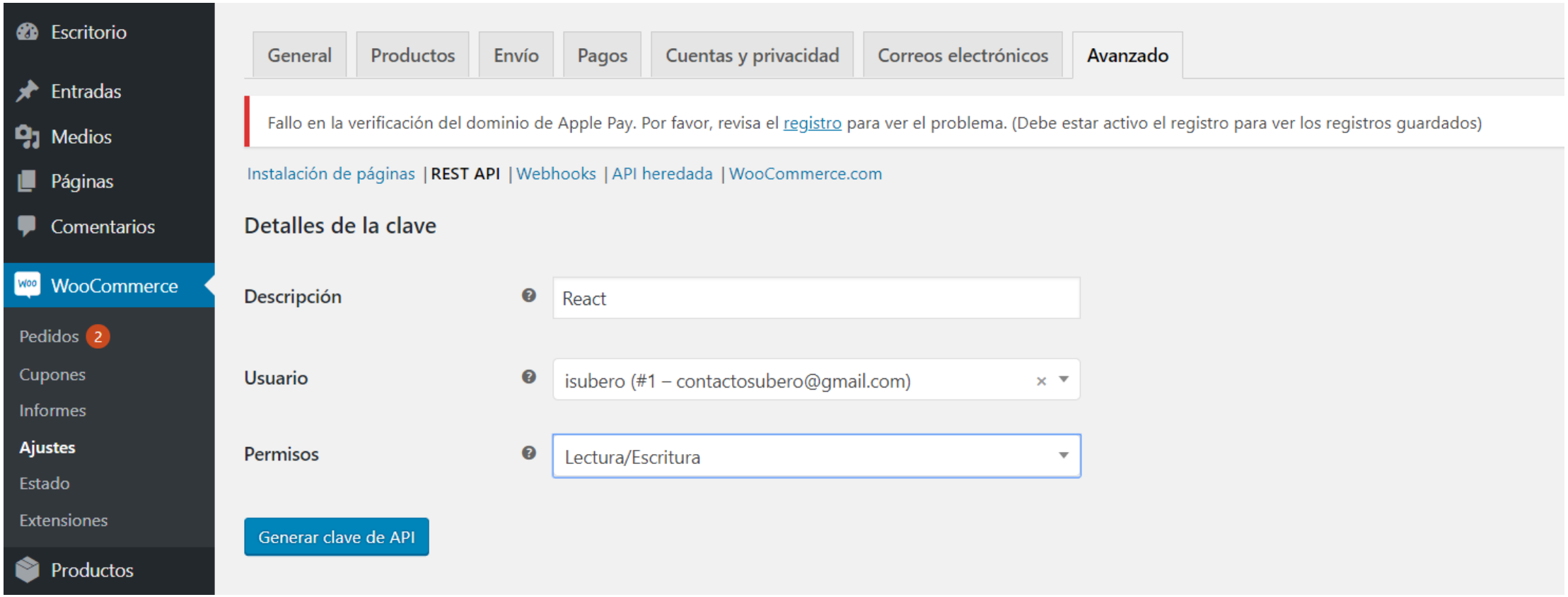
axios

Axios
Cliente de peticiones HTTP.

Estructura de Ficheros de CRA

```
my-app/
  README.md
  node_modules/
  package.json
  public/
    index.html
    favicon.ico
  src/
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
```

Generando las credenciales de la API REST de WooCommerce



Generando las credenciales de la API REST de WooCommerce

Rutas con React Router

```
...
import React, { Component } from 'react';
import { BrowserRouter as Router, Route } from 'react-router-dom';
import Products from './components/Products/Products';
import Product from './components/Product/Product';
import Checkout from './components/Checkout/Checkout';

class App extends Component {
  render() {
    return (
      <Router>
        <Route exact path="/" component={Products} />
        <Route path="/product/:id" component={Product} />
        <Route exact path="/checkout" component={Checkout} />
      </Router>
    );
  }
}

export default App;
```

Componente: Productos

https://github.com/isubero/react-store/blob/master/src/components/Products/Products.jsx

```
class Products extends Component {

    static contextType = myContext;

    render() {
        if ( this.context.state.storeProducts.length > 0 ) {
            const products = this.context.state.storeProducts.map(product => {
                return (
                    <div key={product.id} className={classes.productCard} product={product}>
                        <div className="productImage">
                            <Link to={{
                                pathname: `/product/${product.id}`,
                                state: { product: {...product} }
                            }}>
                                <img className={classes.productImg} src={product.images[0].src} alt={product.images[0].name} width="200" />
                            </Link>
                        </div>
                        <div className="productName">{product.name}</div>
                        <div className="productPrice">{product.price}€</div>
                    </div>
                )
            });

            return <div className={classes.productsList}>{products}</div>;
        } else {
            return 'Loading...';
        }
    }
}

export default Products;
```

Componente: Producto

https://github.com/isubero/react-store/blob/master/src/components/Product/Product.jsx

```
class Product extends Component {

    static contextType = myContext;

    render() {
        return (
            <div className={classes.productPage}>
                <div className={classes.imageCol}>
                    <img className={classes.productImage} src={this.props.location.state.product.images[0].src} alt={this.props.location.state.product.name} />
                </div>
                <div className={classes.infoCol}>
                    <h1 className="productTitle">{this.props.location.state.product.name}</h1>
                    <div className={classes.price}>{this.props.location.state.product.price}<span className={classes.currency}>€</span></div>
                    <div className="productDescription">
                        {this.props.location.state.product.short_description.replace(/(<([^>]+)>)/ig,"")}
                    </div>
                    <button
                        className={classes.addToCartBtn}
                        onClick={ (event) => {this.context.addToCart( this.props.location.state.product )} }
                        >
                        Añadir al carrito
                    </button>
                </div>
            </div>
        );
    }
}

export default Product;
```

Componente: Checkout

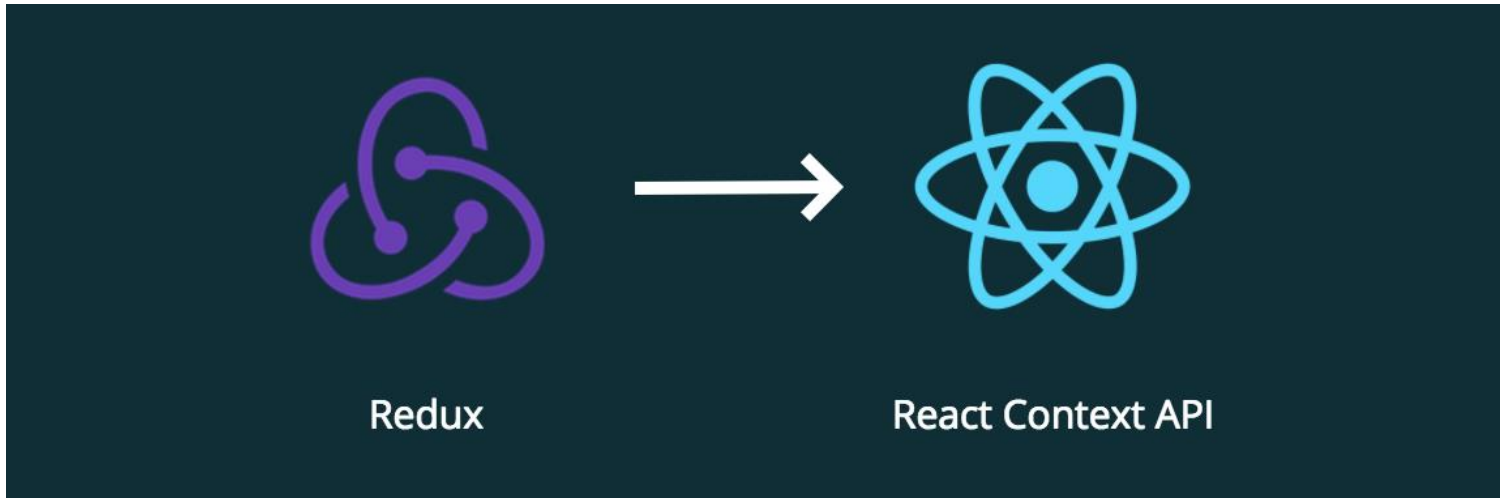
Este componente es demasiado largo como para mostrarlo en capturas de pantallas.

Está disponible en https://github.com/isubero/react-store/blob/master/src/components/Checkout/Checkout.jsx para mostrarlo en detalle durante la presentación.

React Context API

Con React context podemos acceder a datos sin tener que pasarlos explícitamente en cada componente vía props.

No es necesariamente un reemplazo a Redux, pero podría llegar a serlo.



¿Cómo funciona React Context?

- Creamos un contexto con React.createContext()
- createContext() nos devuelve dos componentes, un Provider y un Consumer.
- Le pasamos el valor que queremos que esté disponible a nivel global en la aplicación a nuestro *Provider*.
- Utilizamos el *Consumer* para acceder a esos datos.
- cartContext.js disponible en https://github.com/isubero/react-store/blob/master/src/cartContext.js

Creando nuestro primer Contexto

- Este componente es muy largo para ponerlo en capturas de pantalla.
- Está disponible en https://github.com/isubero/react-store/blob/master/src/cartContext.js para mostrarlo en detalle durante la presentación.

La seguridad lo primero...

```
componentDidMount() {
    // Get products
    axios({
      method: 'get',
      url: 'https://tienda.isaias.io/wp-json/wc/v1/products',
      auth: {
          username: process.env.REACT_APP_WOO_PUBLIC,
          password: process.env.REACT_APP_WOO_SECRET
      }
  })
  .then( response => {
      console.log('Store products:', response.data);
      this.setState({
          storeProducts: response.data
      });

      console.log(response.data);
  } );
}
```

Esto no es seguro. Usar solo durante el desarrollo.

Add to cart

```
// ADD TO CART
addToCart = (product) => {

  // Check if product is already in the cart
  let exists = this.state.items.filter(item => item.id === product.id);

  // If exists update quantity
  if ( exists.length > 0 ) {
      const newState = { ...this.state };
      newState.items.forEach( item => {
          if ( item.id === exists[0].id ) {
              item.quantity = item.quantity + 1;
          }
      });

      newState.subtotal = this.calculateSubtotal();
      this.setState(newState);
      return;
  }

  // If product is NOT in the cart, add it.
  const formattedProduct = {
      id: product.id,
      name: product.name,
      price: parseFloat(product.price),
      image: product.images[0].src,
      quantity: 1
  }

  const newState = {...this.state};
  newState.items.push(formattedProduct);
  newState.subtotal = this.calculateSubtotal();
  this.setState(newState);
}
```

Podemos definir no solo valores, sino métodos como parte del estado global.

Podemos definir el método addToCart() dentro del contexto y hacerlo accesible desde cualquier parte de la app.

Podemos usarlo luego en Product.js para agregar productos al carrito (nuestro contexto global).

Accediendo al contexto: Product.js

```
import React, { Component } from 'react';
import { myContext } from '../../cartContext';
import classes from './Product.module.css';

class Product extends Component {

    static contextType = myContext;

    render() {
        return (
            <div className={classes.productPage}>
                <div className={classes.imageCol}>
                    <img className={classes.productImage} src={this.props.location.state.product.images[0].src} alt={this.props.location.state.product.name} />
                </div>
                <div className={classes.infoCol}>
                    <h1 className="productTitle">{this.props.location.state.product.name}</h1>
                    <div className={classes.price}>{this.props.location.state.product.price}<span className={classes.currency}>€</span></div>
                    <div className="productDescription">
                        {this.props.location.state.product.short_description.replace(/(<([^>]+)>)/ig,"")}
                    </div>
                    <button
                        className={classes.addToCartBtn}
                        onClick={ (event) => {this.context.addToCart( this.props.location.state.product )} }
                        >
                        Añadir al carrito
                    </button>
                </div>
            </div>
        );
    }
}

export default Product;
```

Importamos el contexto

Declaramos el contexto a usar en el componente

Utilizamos el método definido en el contexto.

Accediendo al contexto: Cart.js

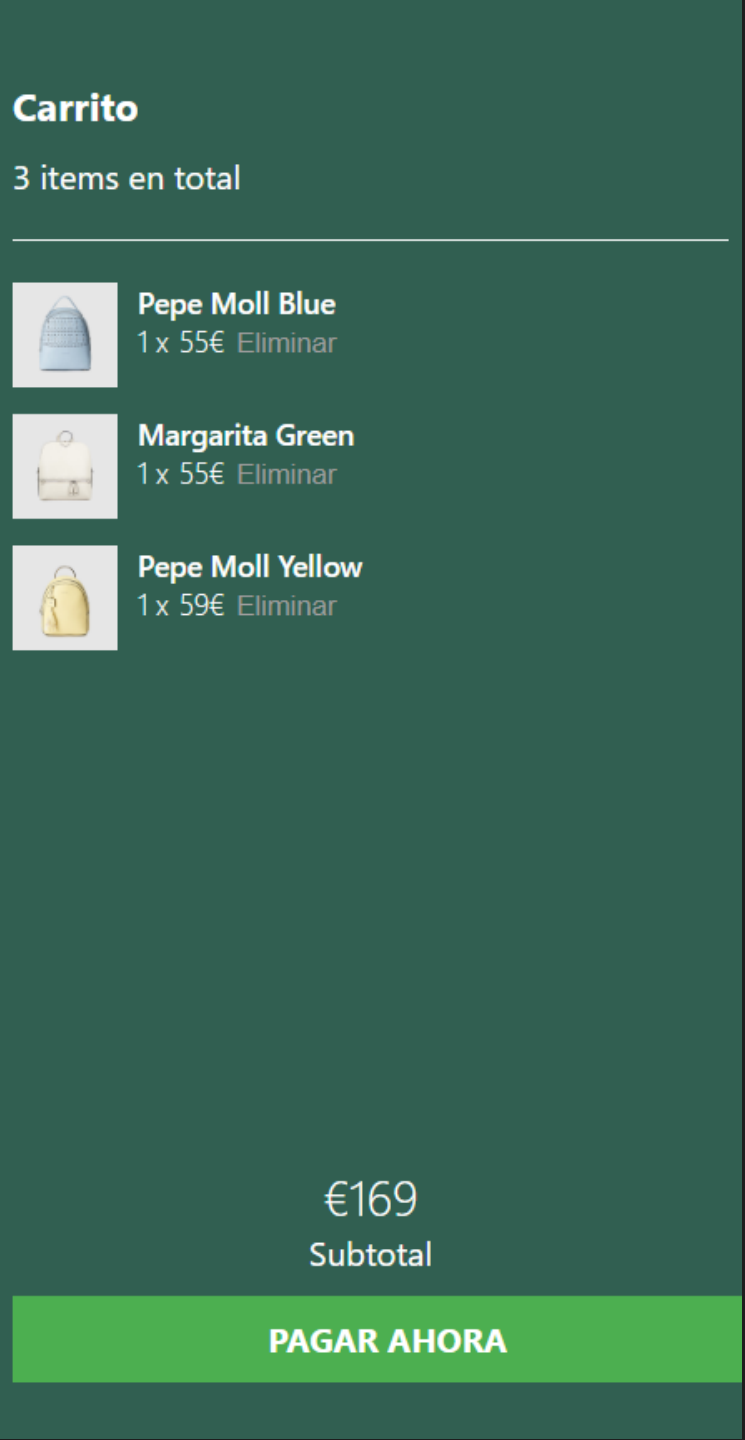
```
render() {

    let cartItems = null;

    if ( this.context.state.items.length > 0 ) {
        cartItems = this.context.state.items.map(item => (
            <CartItem key={item.id} item={item} />
        ))
    } else {
        cartItems = 'Añade un producto al carrito.';
    }

    return (
        <div className={classes.cart}>
            <div className={classes.cartHeader}>
            <h3 className={classes.cartTitle}>Carrito</h3>
            <span className={classes.itemsCount}>{ this.itemsCount() }</span>
            </div>
            <ul className={classes.itemsList}>
                { cartItems }
            </ul>

            <div className={classes.cartFooter}>
                <div className={classes.subtotal}>
                    <div className={classes.subtotalAmount}>€{ this.context.state.subtotal }</div>
                    <span>Subtotal</span>
                </div>
                <Link to="/checkout" className={classes.payBtn}>Pagar ahora</Link>
            </div>
        </div>
    )
}
```



Creando pedidos: Checkout.js

https://github.com/isubero/react-store/blob/master/src/components/Checkout/Checkout.jsx

Atención al método formatLineItems()

Compone los productos y sus cantidades para ser enviados en el formato esperado por la API de WooCommerce.

Atención al método handleSubmit()

Compone los datos de formulario en el formato esperado por la API de WooCommerce.

Finalizar compra

Nombre

Apellido

Teléfono

Correo electrónico

País

España

Dirección de la calle

Nombre de la calle y número de la casa

Apartamento, habitación, etc. (opcional)

Código postal

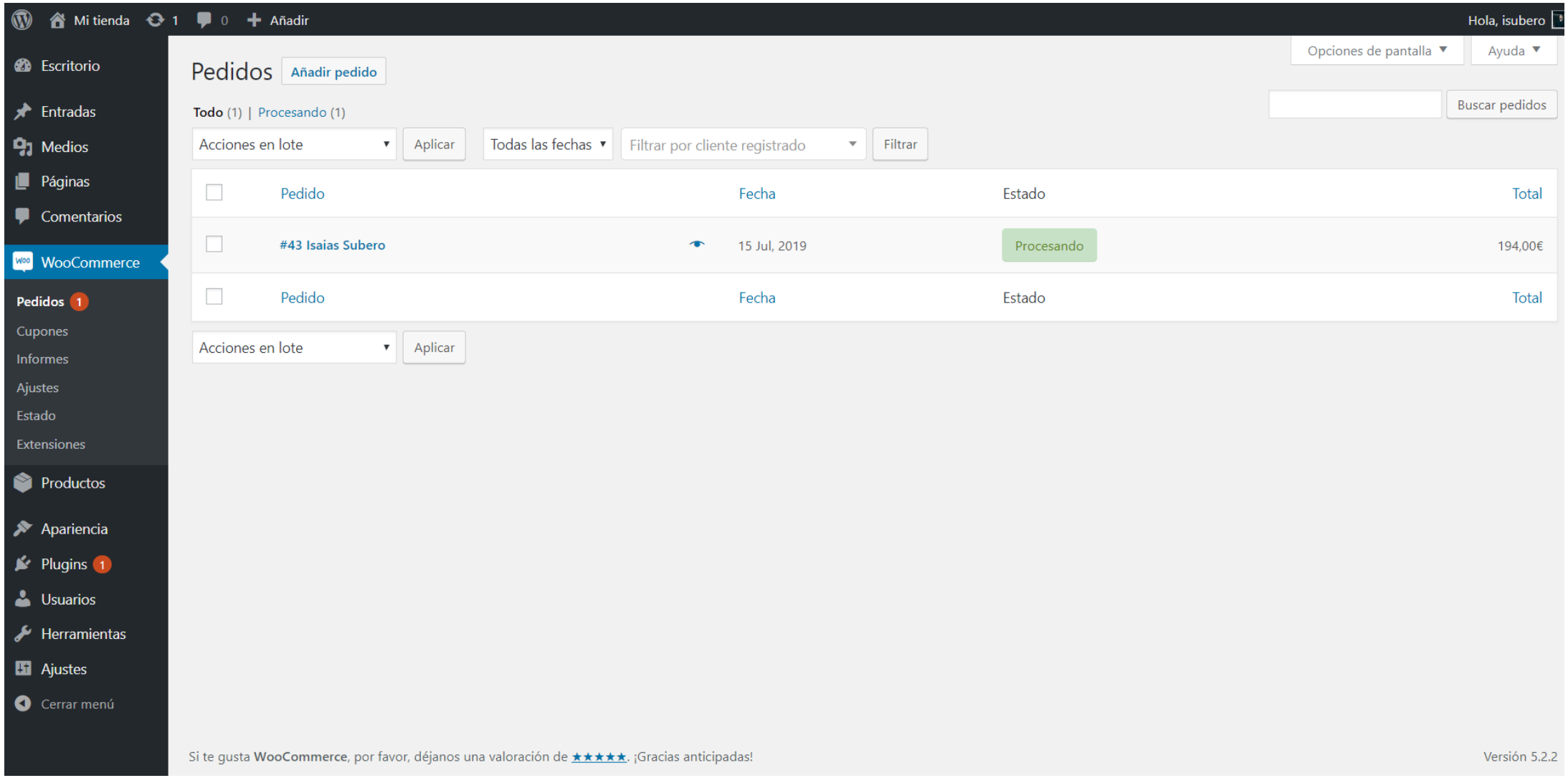
Localidad / Ciudad

Provincia

Asturias

REALIZAR EL PEDIDO

Nuestro pedido creado exitosamente 😄



https://github.com/isubero/react-store

Nos vemos pronto

Isaías Subero
info@isaiassubero.com
Instagram / @isubero
Twitter / @suberovski