

BuyNest e-commerce System: A Comprehensive Overview

IT19143200

This report provides a detailed explanation of e-commerce system architecture, designed to provide multiple user roles and incorporate various functionalities across web and mobile platforms.

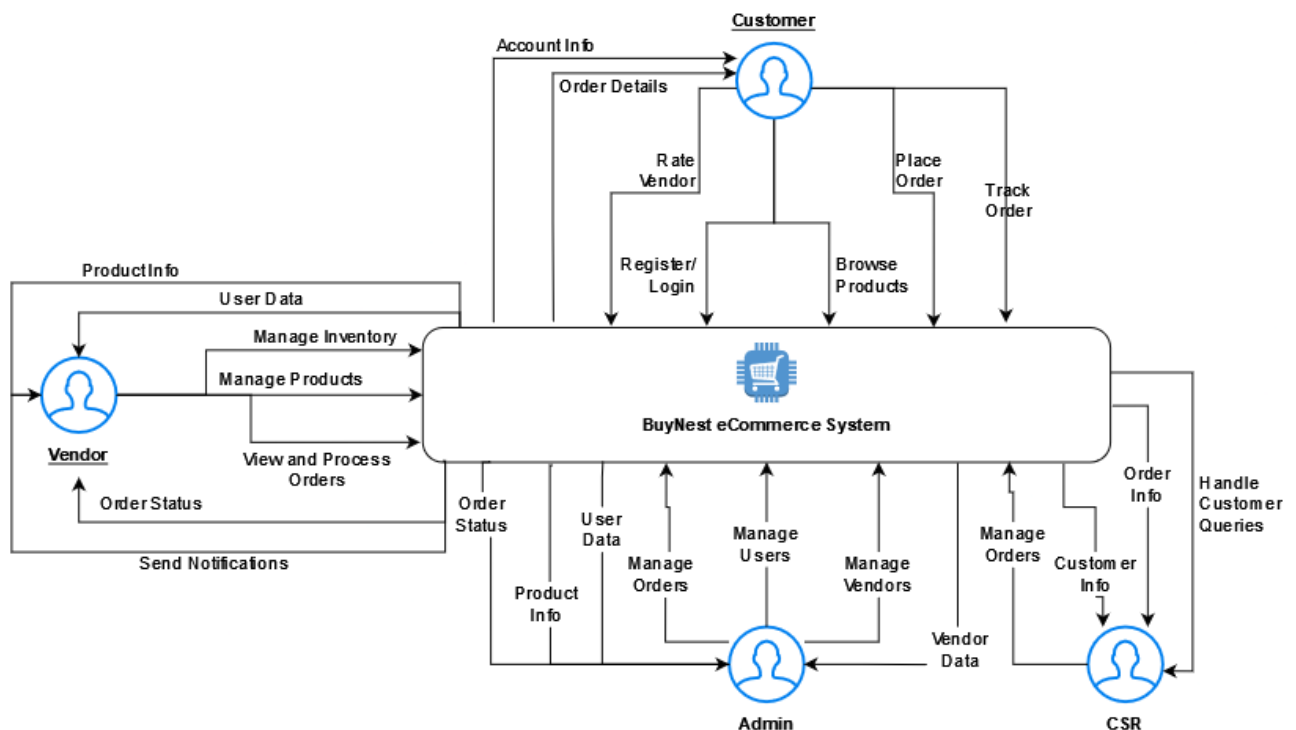
User Roles

Admin: Has full access to all system features and is responsible for creating vendor accounts.

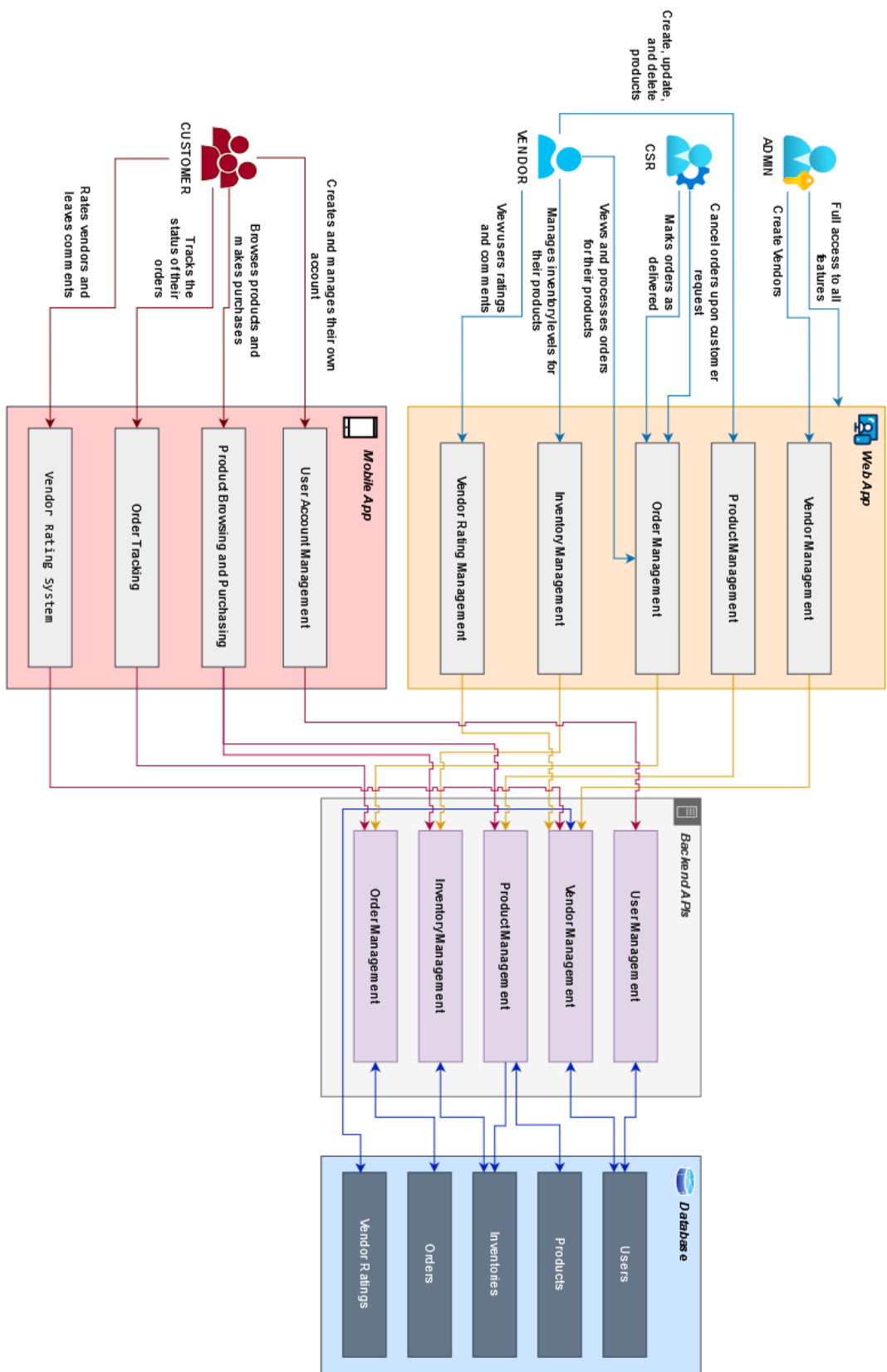
CSR (Customer Service Representative): Manages customer inquiries, including order cancellations and delivery status updates.

Vendor: Manages their products and inventories, views orders, and accesses customer ratings.

Customer: Interacts with the system through the mobile app to browse products, make purchases, track orders, and manage their accounts.



System Overview



Web Application

The web application operates as the primary interface for admin, CSR, and vendor roles.

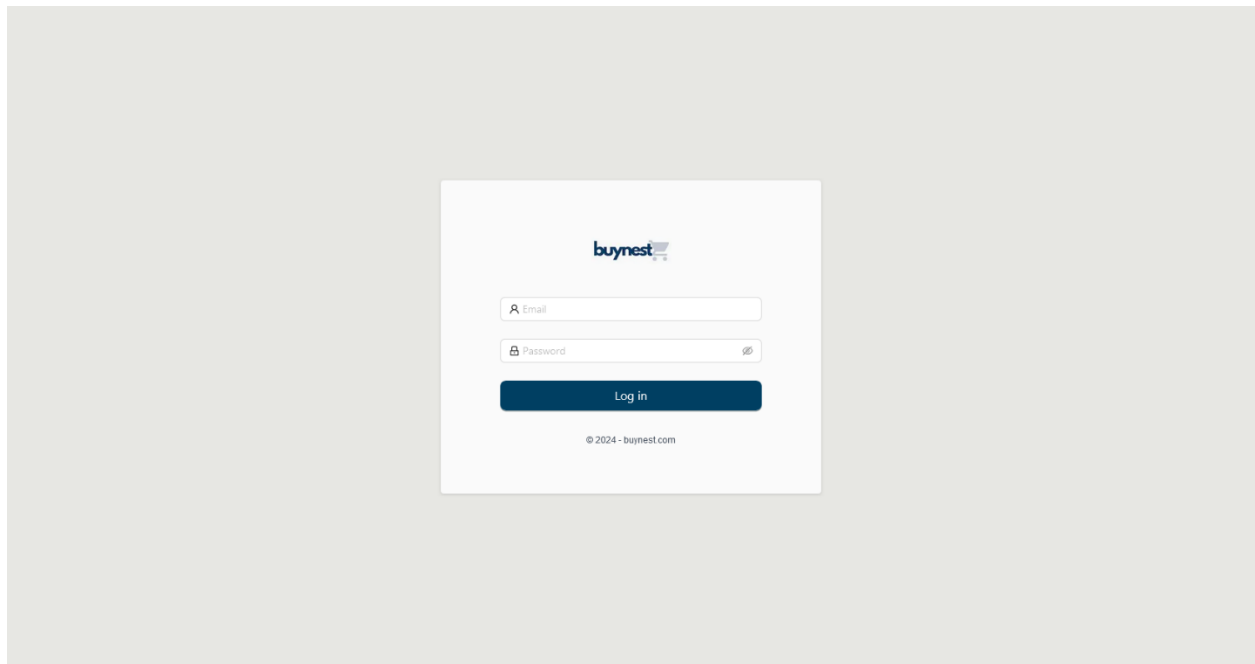
Vendor Management: Enables admins to create and manage vendor accounts.

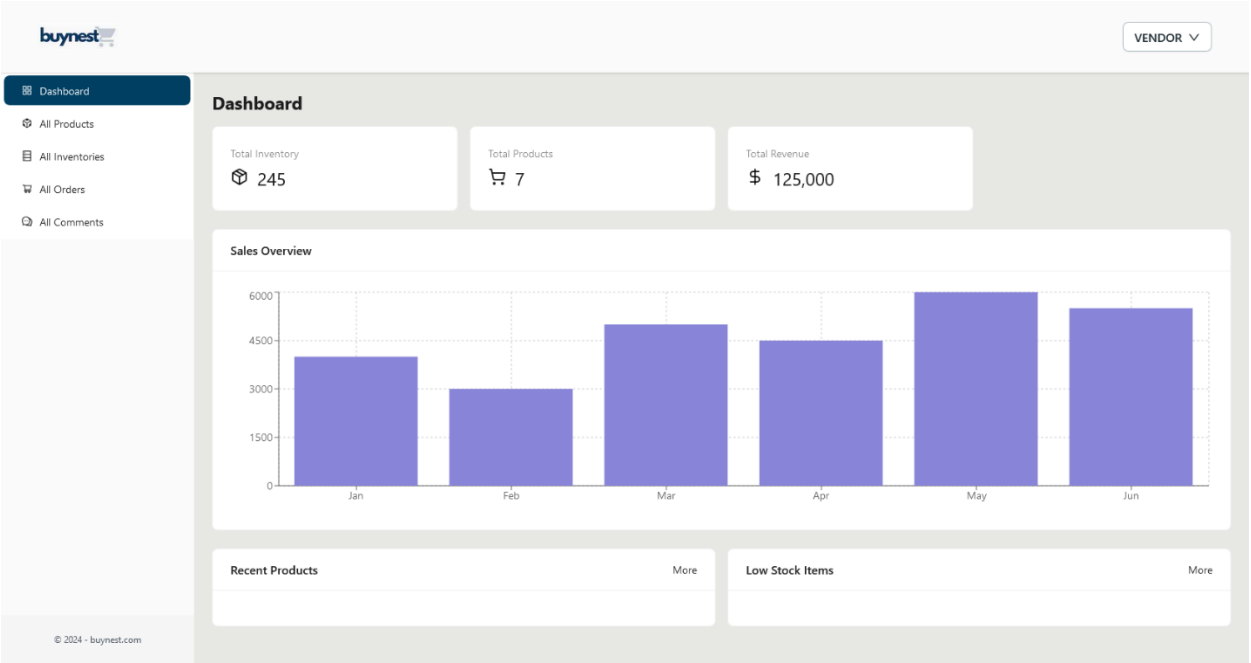
Product Management: Allows vendors to create, update, and delete product listings.

Order Management: Enables order processing and management for CSRs and vendors.

Inventory Management: Enables vendors to track and update their product stock levels.

Vendor Rating Management: Provides vendors with access to customer ratings and feedback.





buynest

VENDOR

Dashboard

All Products

All Inventories

All Orders

All Comments

Product Management

Dashboard > Product Management

ADD NEW PRODUCT

Name	Category	Price	Status	Action
The Great Gatsby by F. Scott Fitzgerald	Books	\$19.99	Active	<div>EditDelete</div>
Life After Life by Kate Atkinson	Books	\$15.00	Active	<div>EditDelete</div>

< 1 >

© 2024 - buynest.com

buynest

VENDOR

Dashboard

All Products

All Inventories

All Orders

All Comments

Inventory Management

Dashboard > Inventory Management

ADD NEW INVENTORY ITEM

Product Name	Category	Price	Quantity	Stock Status	Action
The Great Gatsby by F. Scott Fitzgerald	books	\$19.99	20	In Stock	Update
Life After Life by Kate Atkinson	books	\$15.00	9	Low Stock	Update

< 1 >

© 2024 - buynest.com

buynest

VENDOR

Dashboard

All Products

All Inventories

All Orders

All Comments

All Orders

Dashboard > All Orders

Order ID	Customer	Total Amount	Status	Action
670114d72884ec8e0a849fae	670114d72884ec8e0a849fae	\$39.98	Processing	Update

< 1 >

© 2024 - buynest.com

buynest

VENDOR

Dashboard

All Products

All Inventories

All Orders

All Comments

My Ratings and Comments

Dashboard > My Ratings and Comments

Customer	Rating	Comment	Date	Action
Test 123	★★★★☆	Great service and products!	10/6/2024	View Details
Test 123	★★★★★	Great service and products!	10/6/2024	View Details
Test 123	★★★☆☆	Average service!	10/6/2024	View Details

< 1 >

© 2024 - buynest.com

buynest

VENDOR

Dashboard

All Products

All Inventories

All Orders

All Comments

Inventory Management

Dashboard > Inventory Management

Product Name

The Great Gatsby by F. Scott Fitzgerald

Life After Life by Kate Atkinson

Update Inventory Item

Product Name

Life After Life by Kate Atkinson

Description

Atkinson examines family, history and the power of fiction as she tells the story of a woman born in 1910 – and then tells it again, and again, and

Price

\$ 15

* Quantity

9

* Low Stock Threshold

10

Update

ADD NEW INVENTORY ITEM

Quantity	Stock Status	Action
20	In Stock	Update
9	Low Stock	Update

< 1 >

© 2024 - buynest.com

- Dashboard
- All Products
- All Inventories
- All Orders
- All Comments

Add New Product

Dashboard > Add New Product

* Product Name	* Category
<input type="text"/>	<input type="text" value="Select a category"/>
* Description	* Image URL
<input type="text"/>	<input type="text"/>
* Price	
<input type="text" value="\$"/>	
<input type="button" value="Add Product"/>	<input type="button" value="Reset"/>

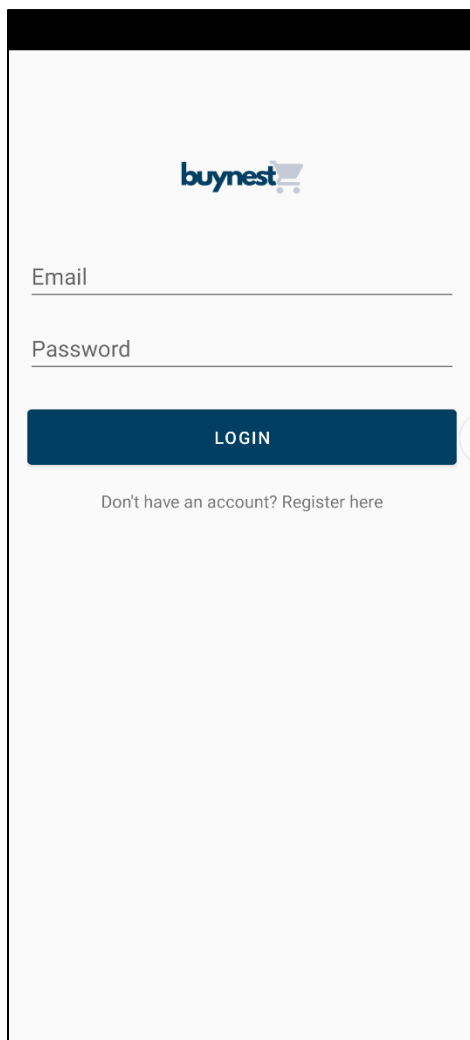
Mobile Application

User Account Management: Allows customers to create and manage their accounts.

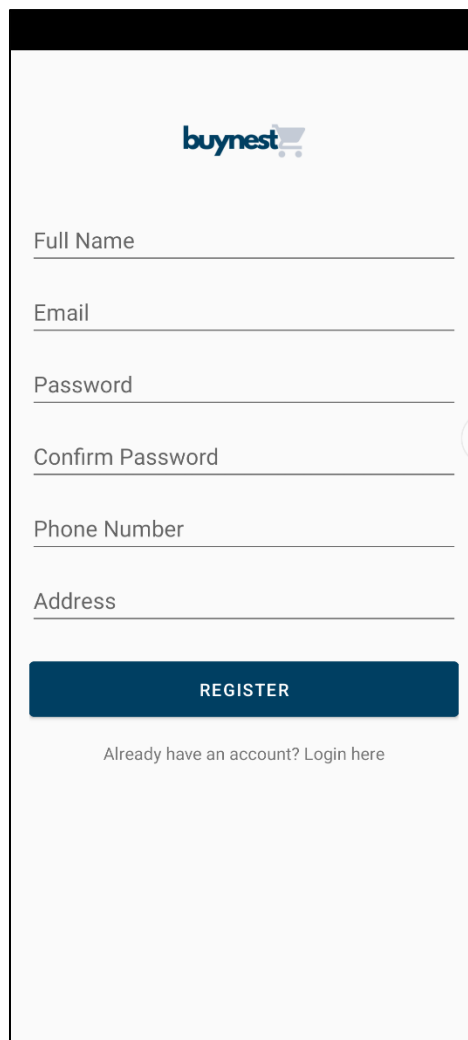
Product Browsing and Purchasing: Enables customers to view products and make purchases.

Order Tracking: This lets customers monitor the status of their orders.

Vendor Rating System: Allows customers to rate vendors and leave comments.



The login screen features a black header bar at the top. Below it, the 'buynest' logo is centered. There are two input fields: 'Email' and 'Password'. A dark blue 'LOGIN' button is positioned below the password field. At the bottom, there is a link that says 'Don't have an account? Register here'.



The registration screen has a black header bar at the top. The 'buynest' logo is centered. It includes five input fields: 'Full Name', 'Email', 'Password', 'Confirm Password', 'Phone Number', and 'Address'. A dark blue 'REGISTER' button is located below the 'Address' field. At the bottom, there is a link that says 'Already have an account? Login here'.



Smartphone X

Vendor:
66f9b50560264073aad7e75f
Category: electronics

\$989.99



Smartphone X Pro

Vendor:
66f9b50560264073aad7e75f
Category: electronics

\$1289.99



Mac Book Pro

Vendor:
66f9b50560264073aad7e75f
Category: electronics

\$1999.00



Mac Book Pro 2

Vendor:
66f9b50560264073aad7e75f
Category: electronics

\$2199.00



test

Vendor:



Home



[Back to Home](#)



The Great Gatsby by F. Scott Fitzgerald

\$19.99

Category: books

Set in the summer of 1922, the novel follows the life of a young and mysterious millionaire, his extravagant lifestyle in Long Island, and his obsessive love for a beautiful former debutante. As the story unfolds, the millionaire's dark secrets and the corrupt reality of the American dream during the Jazz Age are revealed. The narrative is a critique of the hedonistic excess and moral decay of the era, ultimately leading to tragic consequences.

Quantity:

-

1

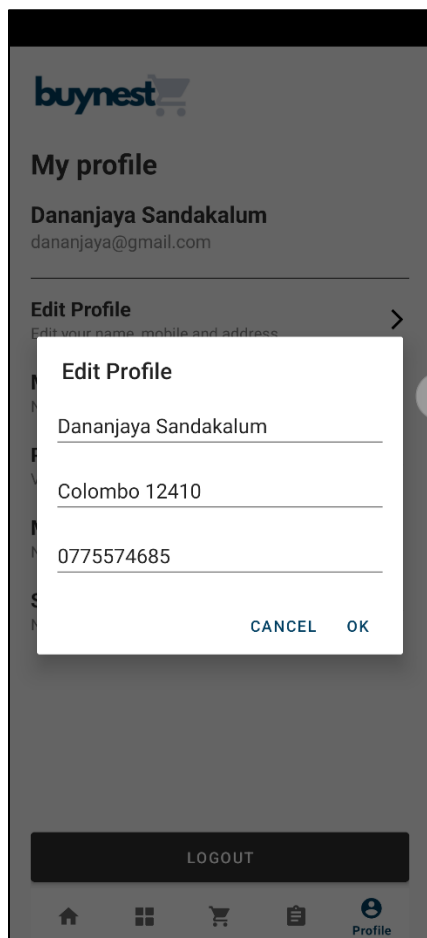
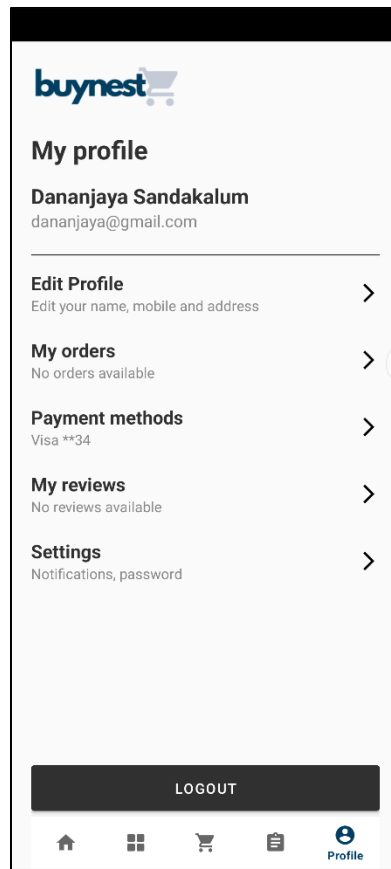
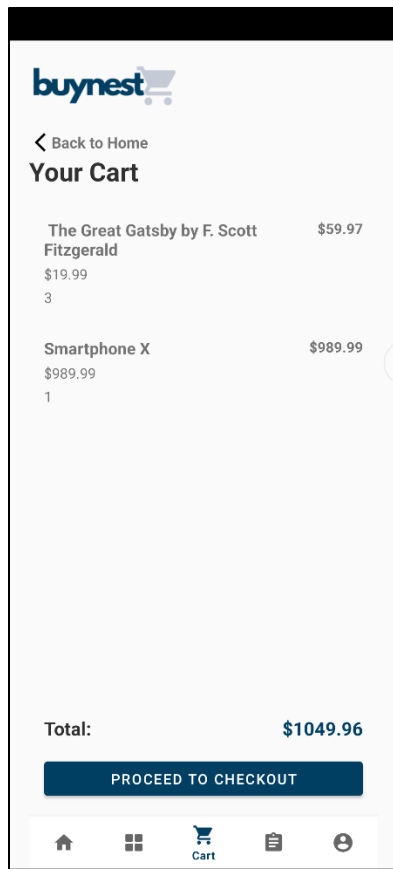
+

ADD TO CART



Home





Backend APIs

User Management: Manages user authentication and authorization across all roles.

Vendor Management: Handles vendor-specific operations and data.

Product Management: Manages product information and listings.

Inventory Management: Tracks and updates product stock levels.

Order Management: Processes and manages customer orders.

```
import express, { json, urlencoded } from "express";
import { connect } from "mongoose";
import { config } from "dotenv";
import cors from "cors";

// Import middlewares
import { authenticateJWT, authorize } from "../middlewares/auth.js";

// Import routes
import userRoutes from "../routes/userRoutes.js";
import productRoutes from "../routes/productRoutes.js";
import orderRoutes from "../routes/orderRoutes.js";
import inventoryRoutes from "../routes/inventoryRoutes.js";
import vendorRoutes from "../routes/vendorRoutes.js";

config();

const app = express();

//middleware
app.use(json());
app.use(urlencoded({ extended: true }));
app.use(cors());

// Routes
app.use("/api/users", userRoutes);
app.use("/api/products", authenticateJWT, productRoutes);
app.use("/api/orders", authenticateJWT, orderRoutes);
app.use("/api/inventory", authenticateJWT, inventoryRoutes);
app.use("/api/vendors", authenticateJWT, vendorRoutes);

connect(process.env.MONGO_URI, {
```

```

    useUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => console.log("Connected to MongoDB"))
  .catch((err) => console.error("MongoDB connection error:", err));

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: "Something went wrong!" });
});

const port = process.env.PORT || 5000;

// Start the server
app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});

export default app;

```

```

import User from "../models/User.js";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";

//Register a user
export const registerUser = async (req, res) => {
  try {
    const { name, email, password, role, address, contact } = req.body;

    if (role === "ADMIN") {
      return res.status(403).json({ message: "Cannot register as admin" });
    }

    const existingUser = await User.findOne({ email: email });
    if (existingUser) {
      return res.status(400).json({ message: "Email already exists" });
    }
  }

```

```

const hashedPassword = await bcrypt.hash(password, 10);
const user = new User({
  email,
  password: hashedPassword,
  name,
  address,
  contact,
  role,
});
await user.save();

res.status(201).json({ message: "User registered successfully" });
} catch (error) {
  res
    .status(500)
    .json({ message: "Error registering user", error: error.message });
}
};

//Login user
export const loginUser = async (req, res) => {
  try {
    const { email, password } = req.body;

    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: "Invalid credentials" });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ message: "Invalid credentials" });
    }

    if (!process.env.JWT_SECRET) {
      throw new Error("JWT_SECRET is not defined in environment variables");
    }

    const token = jwt.sign(
      { id: user._id, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: "1d" }
    );

    res.json({

```

```

    token,
    user: {
      id: user._id,
      username: user.username,
      email: user.email,
      role: user.role,
    },
  });
} catch (error) {
  res.status(500).json({ message: "Error logging in", error: error.message });
}
};

```

//Get user profile

```

export const getUserProfile = async (req, res) => {
  try {
    const user = await User.findById(req.user.id).select("-password");
    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }
    res.json(user);
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error fetching user profile", error: error.message });
  }
};

```

//Update user profile

```

export const updateUserProfile = async (req, res) => {
  try {
    const { name, address, contact } = req.body;

    console.log("update req", req.body);

    const user = await User.findById(req.user.id);
    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }

    if (name) user.name = name;
    if (address) user.address = address;
    if (contact) user.contact = contact;

    await user.save();
  }
};

```

```

    res.json({ message: "Profile updated successfully", user });
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error updating user profile", error: error.message });
  }
};

//Verify the token
export const verifyToken = async (req, res) => {
  try {
    const user = await User.findById(req.user.id).select("-password");

    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }

    res.json({
      message: "Token is valid",
      user: {
        id: user._id,
        username: user.username,
        email: user.email,
        role: user.role,
      },
    });
  } catch (error) {
    console.error("Token verification error:", error);
    res
      .status(500)
      .json({ message: "Error verifying token", error: error.message });
  }
};

```

```

import User from "../models/User.js";
import VendorRating from "../models/VendorRating.js";
import bcrypt from "bcrypt";

//Create new vendor
export const createVendor = async (req, res) => {
  try {
    const { name, email, password, description, address, contact, image } =

```

```

    req.body;

    const existingUser = await User.findOne({ $or: [{ name }, { email }] });
    if (existingUser) {
        return res.status(400).json({ message: "Name or email already exists" });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const vendor = new User({
        name,
        email,
        password: hashedPassword,
        role: "VENDOR",
        description,
        address,
        contact,
        image,
        averageRating: 0,
        totalRatings: 0,
    });
    await vendor.save();

    res.status(201).json({
        message: "Vendor created successfully",
        vendor: {
            id: vendor._id,
            name: vendor.name,
            email: vendor.email,
            description: vendor.description,
            address: vendor.address,
            contact: vendor.contact,
            image: vendor.image,
            averageRating: vendor.averageRating,
            totalRatings: vendor.totalRatings,
        },
    });
} catch (error) {
    res
        .status(500)
        .json({ message: "Error creating vendor", error: error.message });
}
};

//Get all vendors
export const getVendors = async (req, res) => {

```



```

    try {
      const vendors = await User.find({ role: "VENDOR" })
        .select("-password")
        .sort({ _id: -1 });
      res.json(vendors);
    } catch (error) {
      res
        .status(500)
        .json({ message: "Error fetching vendors", error: error.message });
    }
  };

  //Update vendor details
  export const updateVendor = async (req, res) => {
    try {
      const { vendorId } = req.params;
      const { name, email, description, address, contact, image } = req.body;

      const vendor = await User.findById(vendorId);
      if (!vendor || vendor.role !== "VENDOR") {
        return res.status(404).json({ message: "Vendor not found" });
      }

      if (name) vendor.name = name;
      if (email) vendor.email = email;
      if (description) vendor.description = description;
      if (address) vendor.address = address;
      if (contact) vendor.contact = contact;
      if (image) vendor.image = image;

      await vendor.save();
      res.json({ message: "Vendor updated successfully", vendor });
    } catch (error) {
      res
        .status(500)
        .json({ message: "Error updating vendor", error: error.message });
    }
  };

  //Get vendor rating by id
  export const getVendorRatings = async (req, res) => {
    try {
      const { vendorId } = req.params;
      const ratings = await VendorRating.find({ vendorId }).populate(
        "customerId",

```

```

        "name"
    );
    res.json(ratings);
} catch (error) {
    res
        .status(500)
        .json({ message: "Error fetching vendor ratings", error: error.message });
}
};

//Add vendor ratings
export const addVendorRating = async (req, res) => {
    try {
        const { vendorId } = req.params;
        const { rating, comment } = req.body;
        const customerId = req.user.id;

        const vendor = await User.findById(vendorId);
        if (!vendor || vendor.role !== "VENDOR") {
            return res.status(404).json({ message: "Vendor not found" });
        }

        const newRating = new VendorRating({
            vendorId,
            customerId,
            rating,
            comment,
        });
        await newRating.save();

        // Update vendor's average rating and total ratings
        vendor.totalRatings += 1;
        vendor.averageRating =
            (vendor.averageRating * (vendor.totalRatings - 1) + rating) /
            vendor.totalRatings;
        await vendor.save();

        res.status(201).json(newRating);
    } catch (error) {
        res
            .status(500)
            .json({ message: "Error adding vendor rating", error: error.message });
    }
};

```

```

//Delete a vendor by id
export const deleteVendor = async (req, res) => {
  try {
    const { vendorId } = req.params;

    // Check if the user making the request is an admin
    if (req.user.role !== "ADMIN") {
      return res
        .status(403)
        .json({ message: "Only admins can delete vendors" });
    }

    // Find the vendor
    const vendor = await User.findById(vendorId);

    if (!vendor || vendor.role !== "VENDOR") {
      return res.status(404).json({ message: "Vendor not found" });
    }

    // Delete the vendor
    await User.findByIdAndDelete(vendorId);

    // Optionally, delete associated ratings
    await VendorRating.deleteMany({ vendorId });

    res.json({ message: "Vendor deleted successfully" });
  } catch (error) {
    console.error("Error deleting vendor:", error);
    res
      .status(500)
      .json({ message: "Error deleting vendor", error: error.message });
  }
};

//Get total vendor count
export const getTotalVendorCount = async (req, res) => {
  try {
    const totalCount = await User.countDocuments({ role: "VENDOR" });
    res.json({ totalVendorCount: totalCount });
  } catch (error) {
    res.status(500).json({
      message: "Error fetching total vendor count",
      error: error.message,
    });
  }
}

```

```
};
```

```
import Product from "../models/Product.js";
import Inventory from "../models/Inventory.js";

//Create a new product
export const createProduct = async (req, res) => {
  try {
    const { name, description, price, category, image } = req.body;
    const vendorId = req.user.id;

    const product = new Product({
      name,
      description,
      price,
      category,
      image,
      vendorId,
    });
    await product.save();

    // Initialize inventory
    await new Inventory({ productId: product._id, quantity: 0 }).save();

    res.status(201).json(product);
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error creating product", error: error.message });
  }
};

//Get all products
export const getProducts = async (req, res) => {
  try {
    const products = await Product.find({ isActive: true });
    res.json(products);
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error fetching products", error: error.message });
  }
}
```

```

};

//Get products by vendor
export const getProductsByVendor = async (req, res) => {
  try {
    const { vendorId } = req.params;

    // Validate vendorId
    if (!vendorId) {
      return res.status(400).json({ message: "Vendor ID is required" });
    }

    const products = await Product.find({ vendorId, isActive: true });

    if (products.length === 0) {
      return res
        .status(404)
        .json({ message: "No products found for this vendor" });
    }

    res.json(products);
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error fetching products", error: error.message });
  }
};

//Update a product
export const updateProduct = async (req, res) => {
  try {
    const { productId } = req.params;
    const { name, description, price, category } = req.body;

    const product = await Product.findById(productId);
    if (!product) {
      return res.status(404).json({ message: "Product not found" });
    }

    if (
      product.vendorId.toString() !== req.user.id &&
      req.user.role !== "admin"
    ) {
      return res
        .status(403)

```

```

        .json({ message: "Not authorized to update this product" });
    }

    product.name = name;
    product.description = description;
    product.price = price;
    product.category = category;

    await product.save();
    res.json(product);
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error updating product", error: error.message });
  }
};

//Delete a product by id
export const deleteProduct = async (req, res) => {
  try {
    const { productId } = req.params;

    const product = await Product.findById(productId);
    if (!product) {
      return res.status(404).json({ message: "Product not found" });
    }

    if (
      product.vendorId.toString() !== req.user.id &&
      req.user.role !== "admin"
    ) {
      return res
        .status(403)
        .json({ message: "Not authorized to delete this product" });
    }

    await Product.findByIdAndDelete(productId);
    await Inventory.findOneAndDelete({ productId });

    res.json({ message: "Product deleted successfully" });
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error deleting product", error: error.message });
  }
}

```

```

};

//Get total product count
export const getTotalProductCount = async (req, res) => {
  try {
    const totalCount = await Product.countDocuments({ isActive: true });
    res.json({ totalProductCount: totalCount });
  } catch (error) {
    res.status(500).json({
      message: "Error fetching total product count",
      error: error.message,
    });
  }
};

```

```

import Order from "../models/Order.js";
import Product from "../models/Product.js";

//Create a new order
export const createOrder = async (req, res) => {
  try {
    const { products, shippingAddress } = req.body;
    const customerId = req.user.id;

    let totalAmount = 0;
    const orderProducts = [];

    for (let item of products) {
      const product = await Product.findById(item.productId);
      if (!product) {
        return res
          .status(404)
          .json({ message: `Product ${item.productId} not found` });
      }
      totalAmount += product.price * item.quantity;
      orderProducts.push({
        productId: item.productId,
        quantity: item.quantity,
        price: product.price,
      });
    }
  }
};

```

```

const newOrder = new Order({
  customerId,
  products: orderProducts,
  totalAmount,
  shippingAddress,
});

await newOrder.save();
res.status(201).json(newOrder);
} catch (error) {
  res
    .status(500)
    .json({ message: "Error creating order", error: error.message });
}
};

//Get all orders
export const getOrders = async (req, res) => {
  try {
    const orders = await Order.find().populate("customerId", "name email");
    res.json(orders);
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error fetching orders", error: error.message });
  }
};

//Update a order
export const updateOrder = async (req, res) => {
  try {
    const { orderId } = req.params;
    const { shippingAddress, status } = req.body;

    const order = await Order.findById(orderId);
    if (!order) {
      return res.status(404).json({ message: "Order not found" });
    }

    if (shippingAddress) {
      order.shippingAddress = shippingAddress;
    }

    if (status) {
      order.status = status;
    }
  }
};

```



```

    }

    order.updatedAt = Date.now();
    await order.save();

    res.json(order);
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error updating order", error: error.message });
  }
};

//Cancel a order
export const cancelOrder = async (req, res) => {
  try {
    const { orderId } = req.params;

    const order = await Order.findById(orderId);
    if (!order) {
      return res.status(404).json({ message: "Order not found" });
    }

    if (order.status !== "Processing") {
      return res
        .status(400)
        .json({ message: "Can only cancel orders that are still processing" });
    }

    order.status = "Cancelled";
    order.updatedAt = Date.now();
    await order.save();

    res.json({ message: "Order cancelled successfully", order });
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error cancelling order", error: error.message });
  }
};

//Get orders by vendor
export const getOrdersByVendorProducts = async (req, res) => {
  try {
    const { vendorId } = req.params;

```

```

const vendorProducts = await Product.find({ vendorId: vendorId }, "_id");
const vendorProductIds = vendorProducts.map((product) => product._id);

const orders = await Order.find({
  "products.productId": { $in: vendorProductIds },
})
  .populate("customerId", "name email")
  .populate("products.productId", "name price");

if (orders.length === 0) {
  return res
    .status(404)
    .json({ message: "No orders found with products from this vendor" });
}

res.json(orders);
} catch (error) {
  res
    .status(500)
    .json({ message: "Error fetching orders", error: error.message });
}
};

```

```

import Inventory from "../models/Inventory.js";
import Product from "../models/Product.js";
import { sendLowStockNotification } from "../utils/notifications.js";

//Get all inventories
export const getAllInventory = async (req, res) => {
  try {
    const inventoryItems = await Inventory.find()
      .populate({
        path: "productId",
      })
      .exec();
    res.json(inventoryItems);
  } catch (error) {
    res
      .status(500)

```

```

        .json({ message: "Error fetching inventory", error: error.message });
    }
};

//Get inventory by vendor
export const getInventory = async (req, res) => {
    try {
        const { vendorId } = req.params;
        const inventoryItems = await Inventory.find()
            .populate({
                path: "productId",
                match: { vendorId: vendorId },
            })
            .exec();

        const filteredInventory = inventoryItems.filter((item) => item.productId);

        if (filteredInventory.length === 0) {
            return res
                .status(404)
                .json({ message: "No inventory items found for this vendor" });
        }

        res.json(filteredInventory);
    } catch (error) {
        res
            .status(500)
            .json({ message: "Error fetching inventory", error: error.message });
    }
};

//Update the inventory
export const updateInventory = async (req, res) => {
    try {
        console.log("req.params", req.params);
        const { productId } = req.params;
        const { quantity } = req.body;

        const inventory = await Inventory.findOne({ productId: productId });
        if (!inventory) {
            return res.status(404).json({ message: "Inventory not found" });
        }

        inventory.quantity = quantity;
        await inventory.save();
    }
};

```

```

    if (quantity <= inventory.lowStockThreshold) {
      const product = await Product.findById(productId);
      await sendLowStockNotification(product.vendorId, product.name, quantity);
    }

    res.json(inventory);
  } catch (error) {
    res
      .status(500)
      .json({ message: "Error updating inventory", error: error.message });
  }
};

//Get total inventory count
export const getTotalInventoryCount = async (req, res) => {
  try {
    const totalCount = await Inventory.aggregate([
      {
        $group: {
          _id: null,
          totalQuantity: { $sum: "$quantity" },
        },
      },
    ]);

    res.json({ totalInventoryCount: totalCount[0]?.totalQuantity || 0 });
  } catch (error) {
    res.status(500).json({
      message: "Error fetching total inventory count",
      error: error.message,
    });
  }
};

```

NOSQL Database

