# Billing-Site Documentation

## Overview

The Billing-Site application is designed to manage product sales and transactions effectively. It features multiple pages that handle distinct functionalities such as product listing, transaction management, and user interactions.

## Abstract

The Billing-Site project is a comprehensive web application aimed at streamlining the sales and transaction management processes. By integrating features like product categorization, transaction tracking, and user-friendly interfaces, the application provides an efficient and intuitive solution for businesses. The modular structure and dynamic data handling ensure scalability and ease of maintenance, making it a robust tool for modern business needs.

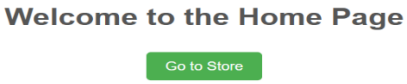
## Home Page

The Home Page is the landing page of the Billing-Site application. It provides an overview of the application and navigation links to other features like product categories and transaction history.

```
import React from 'react';
import { Link } from 'react-router-dom';

const HomePage = () => {
  return (
    <div>
      <h1>Welcome to the Billing-Site</h1>
      <nav>
        <ul>
          <li><Link to="/products">Product Categories</Link></li>
          <li><Link to="/transactions">Transaction History</Link></li>
        </ul>
      </nav>
    </div>
  );
```

```
};
export default HomePage;
```



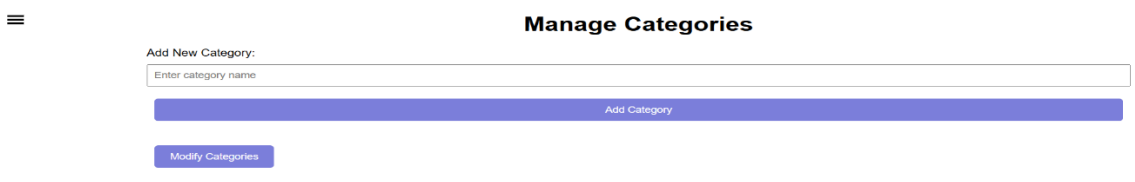


# Billing-Site Documentation

## Product Categories Page

The Product Categories Page displays a list of available product categories. Users can browse and select specific categories to view related products.

```
import React, { useState, useEffect } from 'react';
const ProductCategoriesPage = () => {

  const [categories, setCategories] = useState([]);



  useEffect(() => {

    fetch('/api/categories')

      .then(response => response.json())

      .then(data => setCategories(data));

  }, []);



  return (

    <div>

      <h1>Product Categories</h1>

      <ul>

        {categories.map(category => (

          <li key={category.id}>{category.name}</li>

        ))}

      </ul>

    </div>
  );

};

export default ProductCategoriesPage;
```



# Billing-Site Documentation

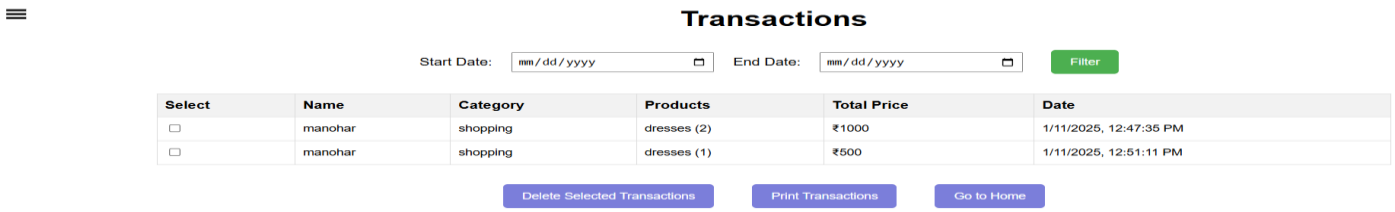
## Transaction History Page

The Transaction History Page allows users to view details of past transactions, including product details, quantities, and amounts.

```
import React, { useState, useEffect } from 'react';
const TransactionHistoryPage = () => {

  const [transactions, setTransactions] = useState([]);
  useEffect(() => {

    fetch('/api/transactions')

      .then(response => response.json())

      .then(data => setTransactions(data));

  }, []);
  return (

    <div>

      <h1>Transaction History</h1>

      <table>

        <thead>

          <tr>

            <th>ID</th>

            <th>Date</th>

            <th>Amount</th>

          </tr>

        </thead>
        <tbody>
          {transactions.map(transaction => (
            <tr key={transaction.id}>
              <td>{transaction.id}</td>
              <td>{transaction.date}</td>
              <td>{transaction.amount}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};
```

**export default TransactionHistoryPage;**

# Billing-Site Documentation



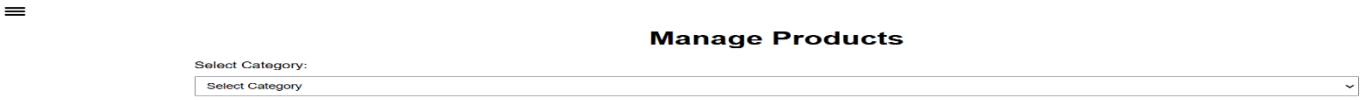

## ADD/EDIT PRODUCT PAGE

This page allows users to add or edit product details, including name, price, and category.

```
import React, { useState } from 'react';

const AddEditProductPage = ({ onSave }) => {
  const [product, setProduct] = useState({ name: '', price: 0, category: '' });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setProduct({ ...product, [name]: value });

  };

  const handleSubmit = (e) => {
    e.preventDefault();
    onSave(product);
  };

  return (
    <div>
      <h1>{product.id ? 'Edit Product' : 'Add Product'}</h1>
      <form onSubmit={handleSubmit}>
        <label>
          Name:
```

```
          <input
            type="text"
            name="name"
            value={product.name}
            onChange={handleChange}
          />
        </label>
        <label>
          Price:
          <input
            type="number"
            name="price"
            value={product.price}
            onChange={handleChange}
          />
        </label>
        <label>
          Category:
          <input

            type="text"
            name="category"
            value={product.category}
            onChange={handleChange}
          />
        </label>
        <button type="submit">Save</button>
      </form>
    </div>
  );
};


export default AddEditProductPage;
BU
```

# Billing-Site Documentation




## Buying Page

```
import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import "./SalesPage.css";

const HomePage = () => {
  const [name, setName] = useState("");
  const [category, setCategory] = useState("");
  const [categories, setCategories] = useState({});
  const [selectedProduct, setSelectedProduct] = useState("");
  const [quantity, setQuantity] = useState("");
  const [products, setProducts] = useState([]);
  const [transactionSaved, setTransactionSaved] = useState(false);

  const navigate = useNavigate();


  useEffect(() => {
    const storedCategories = JSON.parse(localStorage.getItem("categories")) ||
{};
    setCategories(storedCategories);
  }, []);

  const handleAddProduct = () => {
    if (selectedProduct && quantity > 0) {
      const productPrice = categories[category]?.products[selectedProduct] || 0;
      const newProduct = {
        name: selectedProduct,
        price: productPrice,
        quantity,
        total: productPrice * quantity,
      };
      setProducts([...products, newProduct]);
      setSelectedProduct("");
      setQuantity("");
    }
  };

  const handleSaveTransaction = () => {
    if (name && category && products.length > 0) {
      const transaction = {
        name,
        category,
        products,
        date: new Date().toISOString(),
      };
      const transactions = JSON.parse(localStorage.getItem("transactions")) ||
```

```
[];
      transactions.push(transaction);
      localStorage.setItem("transactions", JSON.stringify(transactions));
      setName("");
      setCategory("");
      setProducts([]);
      setTransactionSaved(true);
    }
  };

  const handleClearTransaction = () => {
    setTransactionSaved(false);
  };

  const handleQuantityChange = (e) => {
    const value = e.target.value;
    if (value === "" || /^[0-9]+$/.test(value)) {
      setQuantity(value);
    }
  };

  return (
    <div className="container">
      <h1>Buying Page</h1>
      <div className="form-container">
        <label>Name:</label>
        <input value={name} onChange={(e) => setName(e.target.value)} />

        <label>Category:</label>
        <select value={category} onChange={(e) => setCategory(e.target.value)}>
          <option value="">Select Category</option>
          {Object.keys(categories).map((cat) => (
            <option key={cat} value={cat}>
              {cat}
            </option>
          ))}
        </select>

        <label>Product:</label>
        <select
          value={selectedProduct}
          onChange={(e) => setSelectedProduct(e.target.value)}
          disabled={!category}
        >
          <option value="">Select Product</option>
          {categories[category]?.products &&
            Object.keys(categories[category].products).map((prod) => (
              <option key={prod} value={prod}>
                {prod}
              </option>
            ))}
        </select>

        <label>Quantity:</label>
        <input
          type="number"
          value={quantity}
          min="1"
          onChange={handleQuantityChange}
          disabled={!selectedProduct}
        />
```

```
      </div>

      <div className="button-container">
        <button onClick={handleAddProduct} disabled={!quantity ||
!selectedProduct}>
          Add Product
        </button>
        <button onClick={handleSaveTransaction} disabled={!products.length}>
          Save Transaction
        </button>
      </div>

      {products.length > 0 && !transactionSaved && (
        <div className="products-list">
          <h2>Added Products</h2>
          {products.map((product, index) => (
            <div key={index} className="product-item">
              <p>
                <strong>Product Name:</strong> {product.name}
              </p>
              <p>
                <strong>Quantity:</strong> {product.quantity}
              </p>
              <p>
                <strong>Total Price:</strong> ₹{product.total}
              </p>
            </div>
          ))}
        </div>
      )}

      {transactionSaved && (
        <div className="transaction-saved">
          <h2>Transaction Saved!</h2>
          <button onClick={handleClearTransaction}>Clear Transaction</button>
        </div>
      )}
    </div>
  );
};

export default HomePage;
```

# Billing-Site Documentation

## Buying Page

Name:

Category:

Select Category

Product:

Select Product

Quantity:

Add Product

Save Transaction

## AUTH.JSX PAGE

```
import React, { useState } from "react";
import { useNavigate, Link } from "react-router-dom";
import loginimage from "../assets/login.png";
import signupimage from "../assets/signup.png";
import "./auth.css";

const Auth = () => {
  const [isLogin, setIsLogin] = useState(true);
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [username, setUsername] = useState("");
  const [confirmPassword, setConfirmPassword] = useState("");
  const [errorMessage, setErrorMessage] = useState("");
  const navigate = useNavigate();

  const handleLogin = async () => {
    try {
      const response = await fetch("http://localhost:5000/api/auth/login", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ email, password }),
      });

      const data = await response.json();

      if (response.ok) {
        localStorage.setItem("loggedInUser", JSON.stringify(data.user));
        localStorage.setItem("token", data.token);
        navigate("/");
      } else {
        setErrorMessage(data.message || "Invalid credentials");
        setEmail("");
        setPassword("");
      }
    } catch (error) {
      setErrorMessage("Server error. Please try again later.");
      setEmail("");
      setPassword("");
    }
  };
```

```
  const handleSignup = async () => {
    const response = await fetch("http://localhost:5000/api/auth/signup", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ username, email, password, confirmPassword }),
    });

    const data = await response.json();

    if (response.status === 201) {
      setIsLogin(true);
    } else {
      alert(data.message);
    }
  };

  return (
    <div className="auth-container">
      <div className={`auth-form-container ${isLogin ? 'login-page' : 'signup-page'}`}>
        <div className={`auth-image ${isLogin ? 'login-page' : 'signup-page'}`}></div>
        <div className="auth-form">
          <h2>{isLogin ? "Login" : "Signup"}</h2>
          {errorMessage && <p>{errorMessage}</p>}

          {!isLogin && (
            <input
              type="text"
              placeholder="Username"
              value={username}
              onChange={(e) => setUsername(e.target.value)}
            />
          )}

          <input
            type="email"
            placeholder="Email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
          />
          <input
            type="password"
            placeholder="Password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />

          {!isLogin && (
            <input
              type="password"
              placeholder="Confirm Password"
              value={confirmPassword}
              onChange={(e) => setConfirmPassword(e.target.value)}
            />
          )}

          <button onClick={isLogin ? handleLogin : handleSignup}>
            {isLogin ? "Login" : "Sign Up"}
          </button>

          {isLogin ? (
            <p>
              Don't have an account? <Link to="#" onClick={() => setIsLogin(false)}>Sign up
here</Link>
```

```
            </p>
          ) : (
            <p>
              Already have an account? <Link to="#" onClick={() => setIsLogin(true)}>Login
here</Link>
            </p>
          )}
        </div>
      </div>
    </div>
  );
};

export default Auth;
```

# BACKEND

## DB.JS

```
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI, { useNewUrlParser: true, useUnifiedTopology: true
});
    console.log('MongoDB Connected');
  } catch (error) {
    console.error(error);
    process.exit(1);
  }
};

module.exports = connectDB;
```

## CONTROLLERS

```
const Category = require('../models/Category');

const addCategory = async (req, res) => {
  const { name } = req.body;
  const category = new Category({ name });
  await category.save();
  res.status(201).json(category);
};

const getCategories = async (req, res) => {
  const categories = await Category.find();
  res.status(200).json(categories);
};

module.exports = { addCategory, getCategories };
const Product = require('../models/Product');
const mongoose = require('mongoose');

const addProduct = async (req, res) => {
  try {
    const { name, price, category } = req.body;
    const product = new Product({ name, price, category });
    await product.save();
    res.status(201).json(product);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};

const addProductsBulk = async (req, res) => {
  try {
    if (!Array.isArray(products)) {
      return res.status(400).json({ message: "Input must be an array of products." });
    }
    const result = await Product.insertMany(products);
    res.status(201).json(result);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};
```

# Billing-Site Documentation

```
const getProducts = async (req, res) => {
  try {
    const products = await Product.find().populate('category', 'name');
    res.status(200).json(products);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
const searchProducts = async (req, res) => {
  const { term } = req.query;

  try {
    // Search for products by name only
    const products = await Product.find({
      name: { $regex: term, $options: 'i' },
    });

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

    res.status(200).json(products);
  } catch (error) {
    console.error('Error searching products:', error);
    res.status(500).json({ message: error.message });
  }
};

module.exports = { addProduct, addProductsBulk, getProducts,searchProducts };

const Transaction = require('../models/transactionModel');

const getTransactions = async (req, res) => {
  try {
    const transactions = await Transaction.find();
    res.status(200).json(transactions);
  } catch (error) {
    res.status(500).json({ message: 'Error fetching transactions' });
  }
};

const createTransaction = async (req, res) => {
  try {
    const { name, category, products, totalAmount } = req.body;

    if (!name || !category || !products || !totalAmount) {
      return res.status(400).json({ message: 'All fields are required' });
    }

    const transaction = new Transaction({
      name,
      category,
      products,
      totalAmount,
    });

    const savedTransaction = await transaction.save();
    res.status(201).json(savedTransaction);
  } catch (error) {
    res.status(500).json({ message: 'Error creating transaction' });
  }
};

const deleteTransaction = async (req, res) => {
  try {
    const { id } = req.params;
    const deletedTransaction = await Transaction.findByIdAndDelete(id);

    if (!deletedTransaction) {
      return res.status(404).json({ message: 'Transaction not found' });
    }

    res.status(200).json({ message: 'Transaction deleted successfully' });
  } catch (error) {
```

```
    res.status(500).json({ message: 'Error deleting transaction' });
  }
};

module.exports = {
  getTransactions,
  createTransaction,
  deleteTransaction,
};
```

## MODELS

```
const mongoose = require('mongoose');

const categorySchema = mongoose.Schema({
  name: { type: String, required: true, unique: true },
});

const Category = mongoose.model('Category', categorySchema);

module.exports = Category;

const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: { type: Number, required: true },
  category: { type: mongoose.Schema.Types.ObjectId, ref: 'Category', required: true },  // Ensure
category is an ObjectId
});

const Product = mongoose.model('Product', productSchema);
module.exports = Product;

const mongoose = require('mongoose');

const transactionSchema = new mongoose.Schema({
  name: { type: String, required: true },
  category: { type: String, required: true },
  products: [
    {
      name: { type: String, required: true },
      price: { type: Number, required: true },
      quantity: { type: Number, required: true },
      total: { type: Number, required: true },
    },
  ],
  totalAmount: { type: Number, required: true },
  date: { type: Date, default: Date.now },
});

const Transaction = mongoose.model('Transaction', transactionSchema);
module.exports = Transaction;

const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  this.password = await bcrypt.hash(this.password, 10);
  next();
});

const User = mongoose.model('User', userSchema);
module.exports = User;
```

# Billing-Site Documentation

## ROUTES

```
const express = require('express');
const User = require('../models/User');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const router = express.Router();

router.post('/signup', async (req, res) => {
  const { username, email, password, confirmPassword } = req.body;

  if (password !== confirmPassword) {
    return res.status(400).json({ message: "Passwords don't match" });
  }

  try {
    const userExists = await User.findOne({ email });
    if (userExists) {
      return res.status(400).json({ message: 'User already exists' });
    }

    const user = new User({ username, email, password });
    await user.save();
    res.status(201).json({ message: 'User created successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Server error' });
  }
});

router.post('/login', async (req, res) => {
    const { email, password } = req.body;

    try {

      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' });
      }


      const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, {
        expiresIn: '1h',
      });


      res.status(200).json({ message: 'Login successful', token });
    } catch (error) {
      console.error('Login error:', error);
      res.status(500).json({ message: 'Server error' });
    }
  });


module.exports = router;

const express = require('express');
const { addCategory, getCategories } = require('../controllers/categoryController');
const router = express.Router();

router.get('/', getCategories);

router.post('/', addCategory);
router.post('/', async (req, res) => {
    try {
      const { name } = req.body;
      if (!name) return res.status(400).json({ error: 'Category name is required' });
```

```
      const newCategory = new Category({ name });
      await newCategory.save();
      res.status(201).json(newCategory);
    } catch (err) {
      res.status(500).json({ error: 'Failed to create category' });
    }
  });

module.exports = router;

const express = require('express');
const { addProduct, addProductsBulk, getProducts, searchProducts } =
require('../controllers/productController');
const router = express.Router();

router.post('/', addProduct);
router.post('/', async (req, res) => {
    try {
      const { name, price, category } = req.body;
      if (!name || !price || !category) {
        return res.status(400).json({ error: 'All fields are required' });
      }

      const newProduct = new Product({ name, price, category });
      await newProduct.save();
      res.status(201).json(newProduct);
    } catch (err) {
      res.status(500).json({ error: 'Failed to add product' });
    }
  });
router.get('/', getProducts);
router.post('/bulk', addProductsBulk);
router.get('/search', searchProducts);

module.exports = router;

const express = require('express');
const { getTransactions, createTransaction, deleteTransaction } =
require('../controllers/transactionController');

const router = express.Router();

router.get('/', getTransactions);
router.post('/', createTransaction);
router.delete('/:id', deleteTransaction);

module.exports = router;
```

## Conclusion

The Billing-Site application is an effective solution for managing sales and transactions in a streamlined manner. With features like dynamic product categorization, transaction history tracking, and product management, it provides an end-to-end solution for businesses. The modular and scalable design ensures adaptability to evolving business requirements, making it a valuable tool for operational efficiency.

# Billing-Site Documentation

# Billing-Site Documentation

# Billing-Site Documentation

*Space for Screenshot*