LIST OF TABLES

INDEX

1. COMPANY PROFILE

ThinkNEXT Technologies Private Limited, Mohali (Chandigarh) is an ISO 9001:2008 certified company which deals in software Development, Electronics systems development and CAD/CAM consultancy and it is approved from Ministry of Corporate Affairs and registered under Companies Act 1956.

ThinkNEXT deals in University/College/School ERP Software, University Conferences and Journals Management (www.ptuconferences.ac.in, www.ptujournals.ac.in, www.somme.in), Embedded Products, PLC/SCADA Consultancy, GPS based Vehicle Tracking, TechSmart Classes, Android/iPhone Apps development, Web designing, Web development, Discount Deals, Shopping sites, Project Kart, Bulk SMS, Voice SMS (sms5.thinknext.co.in), Bulk Email, Biometric Time Attendance, Access Control, SEO/SMO (Digital Marketing), Database

Solutions, Payment Gateway Integration, E-Mail Integration, Industrial Training, Corporate Training, Placements etc.

ThinkNEXT Technologies provides IT/Electronics solutions using latest technologies e.g. Smart Card (Contact Type, Contactless), NFC, Biometrics, GPS, Barcode, RFID, SMS, Auto SMS (Shortcode), Android, iPhone, Cloud Computing, Web, Windows and Mobile based technologies

ThinkNEXT is Google Partner for Google Adwords, Bing Ads Accredited, Hubspot Inbound and Email Marketing, Facebook Blueprint Certified, Microsoft Bing Ads Accredited Company.

ThinkNEXT has wide expertise in .NET, Crystal Reports, Java, PHP, Android, iPhone, Databases (Oracle and SQL Server), Web Designing, Networking, IIS, Apache, WAMP Web Server configurations, various RAID Levels etc. ThinkNEXT has its sister concerns/associate partners Zeta Apponomics Private Limited and RBH Solutions (Embedded System Products and Industrial Automation using PLC SCADA).

ThinkNEXT has its own multiple Smart Card printing, smart card encoding and barcode label printing machines to provide better and effective customer support solutions. ThinkNEXT has also setup its own placement consultancy and is having numerous placement partner companies to provide best possible placements in IT/ Electronics industry. ThinkNEXT has its numerous clients across the globe. ThinkNEXT has also its offices in USA, Canada, New Delhi, Shimla and Bathinda.

ThinkNEXT Technologies has developed its own cloud computing-based Cloud Campus 4.0 to facilitate knowledge and placement centric services. It is a unique concept for effective and collaborative learning. ThinkNEXT Cloud Campus is a step towards not only 100% placements, but also better job offers even after placements.

ThinkNEXT Value Added Activities:

•Listing of ThinkNEXT in Ministry of Corporate Affairs,Government of India:

Corporate Identity No.: U72200PB2011PTC035677

Status : Approved

- Listing of company in Excise and Taxation, Punjab

TIN No. : 03362166544

•Listing of company in Central Board of Excise and Customs, Ministry of Finance

Service Tax No. : AAECT1486GSD003

- Listing of Company for ESIC

ESIC No. : 12000621820000911

- Technological Collaborations:

Sys build Technologies Pvt. Ltd., Bangalore

- FoxBase Technologies Private Limited, Bangalore
- Enterprise Software Solutions Lab, Bangalore

- Lipidata Systems Limited, Mumbai
- Interworld Commnet, Mohali
- Intersoft Professional, Chandigarh
- Urgent Engineering, Chandigarh
- Ess Dee Engineers, Mohali
- Primary Estates, Mohali
- Bajwa Developers Limited, Mohali
- Authorized dealer for security systems with ADI, ESSL and Base Systems Pvt. Ltd.

- Sister Concerns/Associate Partners
  - ThinkNEXT Smart School, Maur Mandi (Bathinda)
  - Brilliant ITI, Mansa
  - Zeta Apponomics Private Limited, Mohali
  - RBH Solutions, Patiala, Noida

ThinkNEXT Industrial Training Programs under Digital India Scheme (ESDM) and PMKVY 2.0

As ThinkNEXT is also an accredited training partner for Digital India Government Scheme (ESDM) and PMKVY 2.0, Therefore under this scheme, ThinkNEXT offers Free 6 Months industrial training in following programs:

1.Telecom Technician – PC Hardware and Networking

2. Embedded Systems

3. PLC/SCADA (Advanced)

4. Computer Hardware

5. Junior Software Developer

6. Computer Networking and Storage

In this, Dual Certification will be provided to students i.e., ThinkNEXT and Government of India. Students will also be provided National Skill Certificates approved from 5 Government bodies.

ThinkNEXT Industrial Training Programs

CSE/IT/MCA:

1. SAP (ABAP)

2. PHP

3. Android

4. Java

5. SAP (ABAP, MM, PP, SD, HR)

6. .Net

7. Web Designing

8. Professional Hardware, Networking, CCNA, CCNP (With Routers and Managed

Switches)

9. Software Testing

10. Digital Marketing

Electronics/Electrical:

1. SAP (ABAP, MM, PP)

2. Embedded Systems

3. PLC/SCADA (Industrial Automation)

4. Professional Hardware, Networking, CCNA (With Routers and Managed Switches)

5. Android

Mechanical:

1. SAP (MM, PP)
2. AutoCAD
3. Solidworks
4. CNC Programming
5. Solidcam/Delcam/Mastercam
6. CATIA
7. CREO
8. ANSYS
9. NX Unigraphics

Mechanical Industry Tie-ups:

1. Ess Dee Engineers
2. Urgent Engineering
3. 3D Technologies Private Limited

Civil/Architecture:

1. SAP (MM)
2. AutoCAD
3. STAADPro
4. 3DS Max
5. Revit
6. Primavera

Civil Companies Tie-ups:

1. Bajwa Developers Limited
2. TDI Group
3. JLPL Group

Clients:

Some of our prestigious software clients for various ThinkNEXT products/service are:

1. Coromandel International Limited, Secundrabad
2. Medzel, USA
3. Nature9 Inc., USA

4. Punjab Technical University, Jalandhar

5. Maharaja Ranjit Singh Punjab Technical University, Bathinda

6. Guru Kashi University, Talwandi Sabo

7. Rayat Group of Institutions, Ropar

8. Aryans Group of Institutions, Rajpura

9. Punjabi University Patiala

10. Bhai Gurdas Group of Institutions

11. Baba Farid Group of Institutions, Bathinda

12. SUS Group of Institutions, Tangori

13. Asra Group of Institutions, Sangrur

14. Yadavindra College of Engineering and Technology, Talwandi Sabo

17. St. Xavier School, Mansa

And many others

2. INTRODUCTION:

E-commerce is fast gaining ground as an accepted and used business paradigm. More and more business houses are implementing web sites providing functionality for performing commercial transactions over the web. It is reasonable to say that the process of shopping on the web is becoming commonplace. The objective of this

project is to develop a general purpose e-commerce store where product like clothes can be bought from the comfort of home through the Internet. However, for implementation purposes, this paper will deal with an online shopping for clothes. An online store is a virtual store on the Internet where customers can browse the catalog and select products of interest. The selected items may be collected in a shopping cart. At checkout time, the items in the shopping cart will be presented as an order. At that time, more information will be needed to complete the transaction. Usually, the customer will be asked to fill or select a billing address, a shipping address, a shipping option, and payment information such as credit card number. An e-mail notification is sent to the customer as soon as the order is placed E-commerce is fast gaining ground as an accepted and used business paradigm. More and more business houses are implementing web sites providing functionality for performing commercial transactions over the web. It is reasonable to say that the process of shopping on the web is becoming commonplace. The objective of this project is to develop a general purpose e-commerce store where product like clothes can be bought from the comfort of home through the Internet. However, for implementation purposes, this paper will deal with an online shopping for clothes. An online store is a virtual store on the Internet where customers can browse the catalog and select products of interest. The selected items may be collected in a shopping cart. At checkout time, the items in the shopping cart will be presented as an order. At that time, more information will be needed to complete the transaction. Usually, the

customer will be asked to fill or select a billing address, a shipping address, a shipping option, and payment information such as credit card number.

Tableau 1: Product problem assigned

Task No.	Summary	Task description
1	Data Visualization and analysis	Both admin and cashier visualize all the operation and transaction done weekly, analyze the business profit and loss with the help of a dashboard which summarize the number and receipt of transaction, income and products sold.
2	User management	This task can be only done by the admin. It includes : ● creating, editing and deleting user by giving user full name, email address, role (either cahier or admin), username and password, ● Searching user by any keyword or by role, ● And sorting all user by date of creation (newest or oldest), by name and by last active status.

3	Product's category management	Can be only done by the admin, this task includes: • Creating, editing and deleting a product's category (category name), • Searching and sorting a product's category by date of creation (newest or oldest) and by name Many products can belong to one product's category.
4	Product management	admin can execute this task, it includes: • Creating, editing and deleting a product by giving name, price, category, description and image, • Searching and sorting a product by date of creation (newest or oldest) and by name, • And View product details
5	Transaction	Both admin and cashier can execute this task, it includes: • Adding and deleting an item in the cart

8	Account setting	Admin and cashier can update their account details by editing their information and data like their full name, email address, username and password.

3. REQUIREMENT ANALYSIS

Overview

In software development cycle, requirement analysis is the process of defining the expectations of the users for an application that is to be built or modified.

3.1. Problem analysis

Product management

It supports the following functions:

When the admin or cashier wants to add new product, he clicks on “add new data” button, fill the product’s name, price, image, select and hint submit.

The new added product will be stored in the backend database system and a list of all registered category’s product will be generated and displayed in the user’s interface.

The admin can also:

- edit and delete a product by clicking respectively on the edit and delete option in the drop-down menu in every product in the list.
- List and sort product based on the category name, or by date of creation (newest and latest).

User management

A user must log in to buy any product. While logging in, the system will interact to the backend system which will process the user's identification, by comparing the entered credential details with those registered in the database system. While verification is successful, the user will logged in.

3.2. Requirement analysis

As an Admin, the user would like to:

Tableau 2: Requirement analysis (admin)

Requirement ID	Description
F1	Manage user
F1.1	Create new user, modify and remove user
F1.2	Define user right or role
F1.3	Handle login
F1.4	Handle account
F1.4.1	Edit account's details like full name, email address, username and password.
F3	Manage products
F3.1	Add, edit and delete a product
F3.2	Sort products by date of creation or by name
F3.3	Retrieve a specific product by enter a keyword or its name
F3.4	View all registered products category
F4	Handle sale transaction
F4.1	Add or removing item in the cart

As a cashier, the user would like to:

Tableau 3: Requirement analysis (cashier)

Requirement ID	Description
F1	Manage products
F1.1	Add, edit and delete a product
F1.2	Sort products by date of creation or by name
F1.3	Retrieve a specific product by enter a keyword or its name
F1.4	View all registered products category
F2	Handle sale transaction
F2.1	Add or removing item in the cart
F4	Handle account

4. OVERALL DESCRIPTION:

4.1 General Description of project

2.1Description:

- Any member can register and view available products.
- Only registered member can purchase multiple products regardless of quantity.
- Contact Us page is available to contact Admin for queries.
- There are three roles available: Visitor, User and Admin.
    - Visitor can view available products.

- User can view and purchase products.
- An Admin has some extra privilege including all privilege of visitor and user.
- Admin can add products, edit product information and add/remove product.
- Admin can add user, edit user information and can remove user.
- Admin can ship order to user based on order placed by sending confirmation mail.

2. Specific requirements

User interfaces

The user interface will be implemented using the ReactJS Library.

The application GUI will provide menu, buttons, containers, list, grids, forms allowing for easy control by a keyboard and a mouse.

When the user opens the application for the first time, the first showing page must be the dashboard page.

If the user is logged in successfully, he/she should be able to purchase the products .

In every page, except dashboard, only 20 out of all data should be showed and listed (pagination) and a link that allow the user to navigate through the all data. Besides, a search and sort option in which the user can choose the type of search and sort she/he wants to conduct must be implemented.

Every user should have an account page where they can edit their full name, email address, username and password.

The application GUI should vary respective to the user's role and access, i.e. the user with "admin" as a role can access to all menu while whom with "cashier' has a limited menu.

Hardware Interfaces

- A browser that supports JavaScript and HTML and a react developer's tools.

Software Interfaces

The system will utilize Rest API software intermediary to ensure the communication between the user interface application and the mongoDB database store.

The application allows the user to import image via button

The application allows the user to export a report data to a Microsoft excel sheet document.

The application stores data in JSON format to enable easy integrity with third party applications and mongoDB server.

System interfaces

The application runs in:

- the latest version of chrome, opera and firefox
- in Edge 13+, IE 11+, safari and IOS 99+,

4.3 FUNCTIONAL REQUIREMENTS

Manage User

Tableau 5: F1.1. Create new user

Name	Create new user
Description	Done only by the system administrator and she/he must be logged in
Input	User's full name, email ID, username and password
output	List off all registered users including the new created user
Action	Enter user's information, press the submit button
Precondition	All the users' information (mentioned in above Input) must be filled
Postcondition	User's information available

Tableau 6: F1.2. Delete user

Name	Delete user
Description	Done only by the system administrator and she/he must be logged in
Input	User's ID
output	List off all registered users after deletion action
Action	Find the users to be deleted, confirm delete action
Precondition	User must be existed
Postcondition	User deleted

Tableau 7: F1.3. Edit user

Name	Edit user
Description	Done only by the system administrator and she/he must be logged in
Input	User's ID, User's full name, email ID, username and password
output	List off all registered users after editing action
Action	Find the users to be edited, modify the desired information and confirm edit action
Precondition	User must be existed
Postcondition	User edited

Handle transaction

Tableau 8: F2.1. Start new transaction

Name	start new sale transaction
Description	The cashier
Input	Product's details (name, price) and number of products to purchase
output	Amount to be paid
Action	When the customer arrives to the counter, the cashier consults the product's catalog, select the wanted products with its quantity, and the system automatically generates the amount to be paid after calculating the discount if any has been specified in the store setting
Precondition	Products available
Postcondition	Amount shown in dashboard screen (after new sale transaction completed) = amount (before transaction + amount that have been paid

Change account setting

Tableau 11: F4: Change account setting

Name	Change account setting
Description	The cashier
Input	User's full name, email ID, username and password
output	
Action	User login in, select the account menu, modify the information he/she wants to, then press submit
Precondition	User must be registered and logged in successfully
Postcondition	

Manage product

Tableau 12: F5.1. Add new product

Name	Add new product
Description	The cashier
Input	Product's details like name, price, category it belongs to and its picture (optional)
output	List of all available products
Action	Click on add new data button, fill all the required field, select product's category, and press on submit button
Precondition	
Postcondition	List of all available products (after adding action) = list of all available products (before adding action) + the new entered product

Tableau 13: F5.2. Edit product

Name	Edit product
Description	The cashier
Input	Product's ID
output	List of all available products
Action	Select the product to be edited among the listed products, choose edit option in the dropdown list, modify the data and confirm it by pressing submit button
Precondition	Product exist
Postcondition	product with specified ID (after editing) = product with the same ID (before editing)

Tableau 14: F5.3. Delete product

Name	delete product
Description	The cashier
Input	Product's ID
output	List of all available products
Action	Select the product to be deleted among the listed products, choose delete option in the dropdown list, and confirm it.
Precondition	Product exist
Postcondition	List of all available products (after adding action) = list of all available products (before adding action) - the deleted product

Tableau 15: F5.4. Sort product

Name	sort product
Description	The cashier
Input	Product's name or product's date of creation (newest or oldest)
output	List of all sorted products
Action	Select the sorting option by choosing between "sort by name" or sort by date of creation
Precondition	Product exist
Postcondition	

Tableau 16: F5.5. Search product

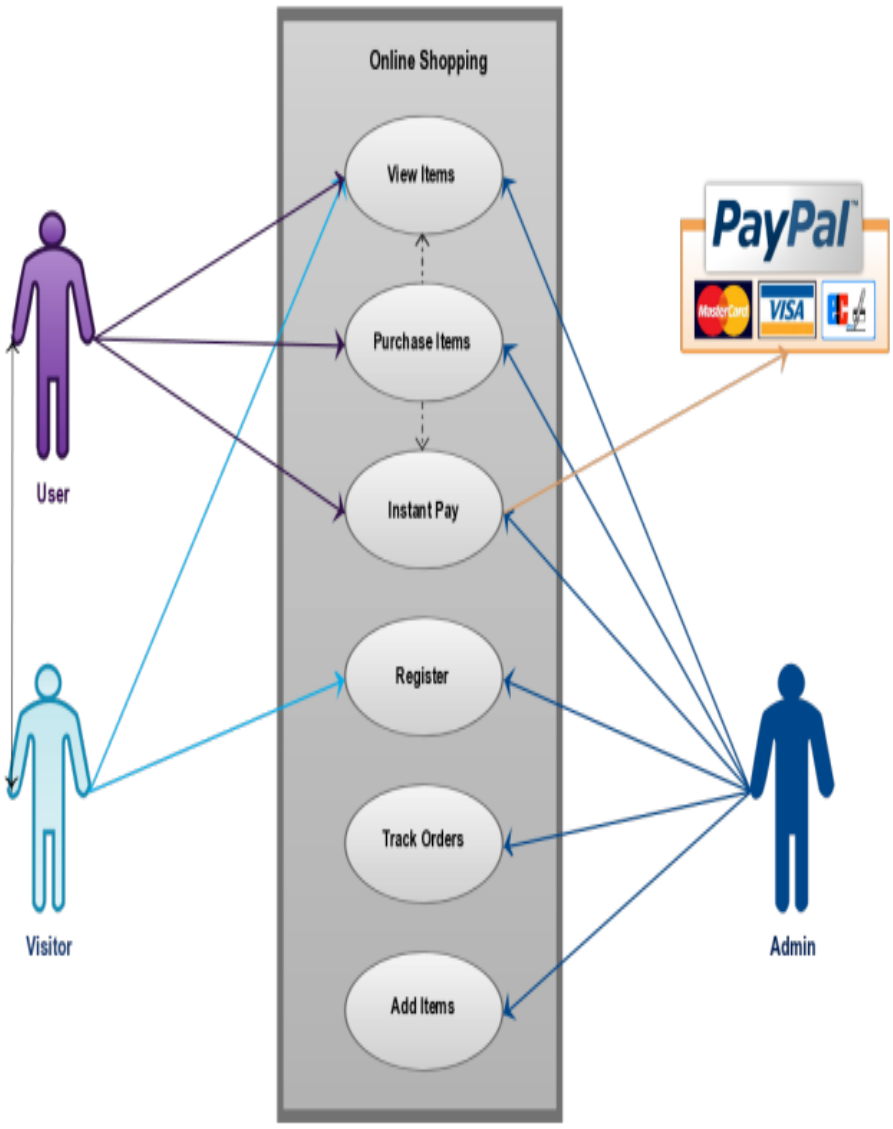
Name	search product
Description	The cashier
Input	Keyword
output	List of matched products
Action	Enter any keyword in the input search field, and press on search button
Precondition	Product exist

Software requirement

- Code editor – Visual studio Code
- Programming Language – JavaScript
- Technology used- MERN stack
- Runtime environment Node.js
- JavaScript GUI Framework & Libraries – ReactJs, Material UI
- Database Management System – NoSQL database MongoDB
- Testing Framework – Junit
- Source Control – Git with GitHub, Trello

- Build Tool – Webpack and Babel

5. User Interface Design



5.1 User case diagram

The global use case diagram of the system is showed by the following figure:

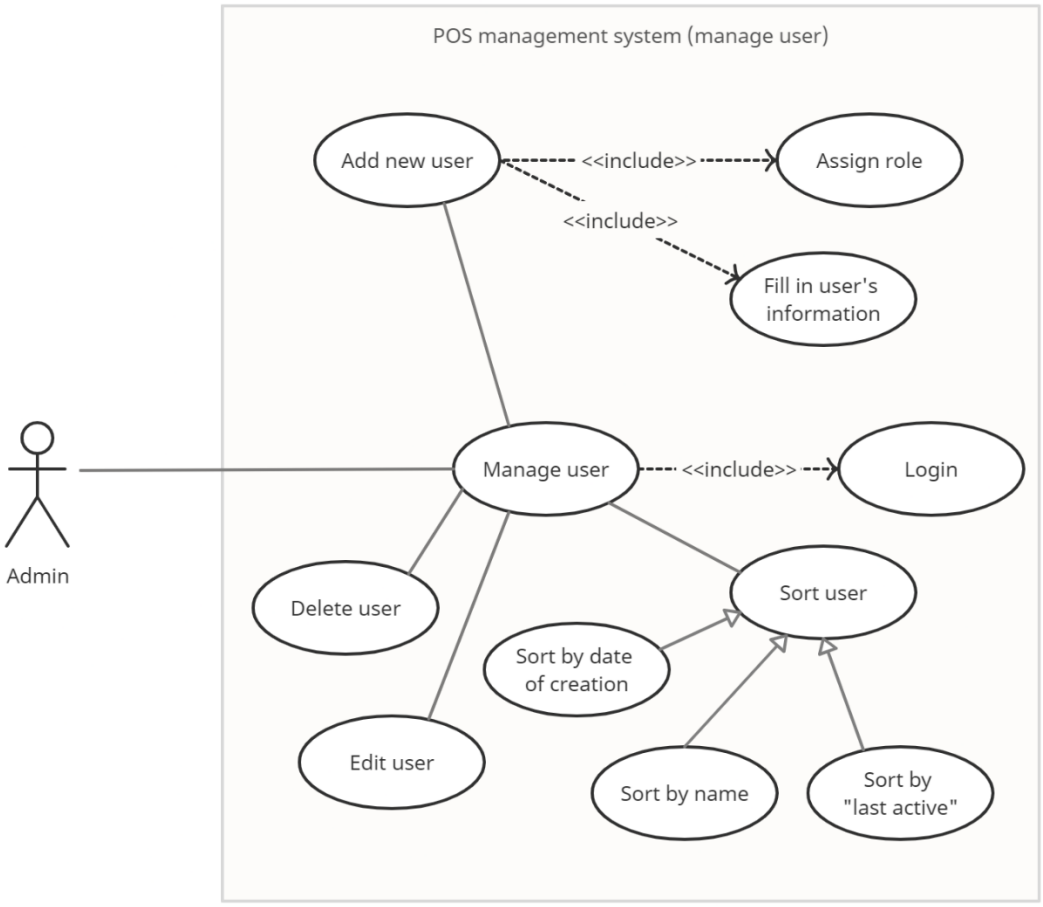


Figure 22: User management use case diagram

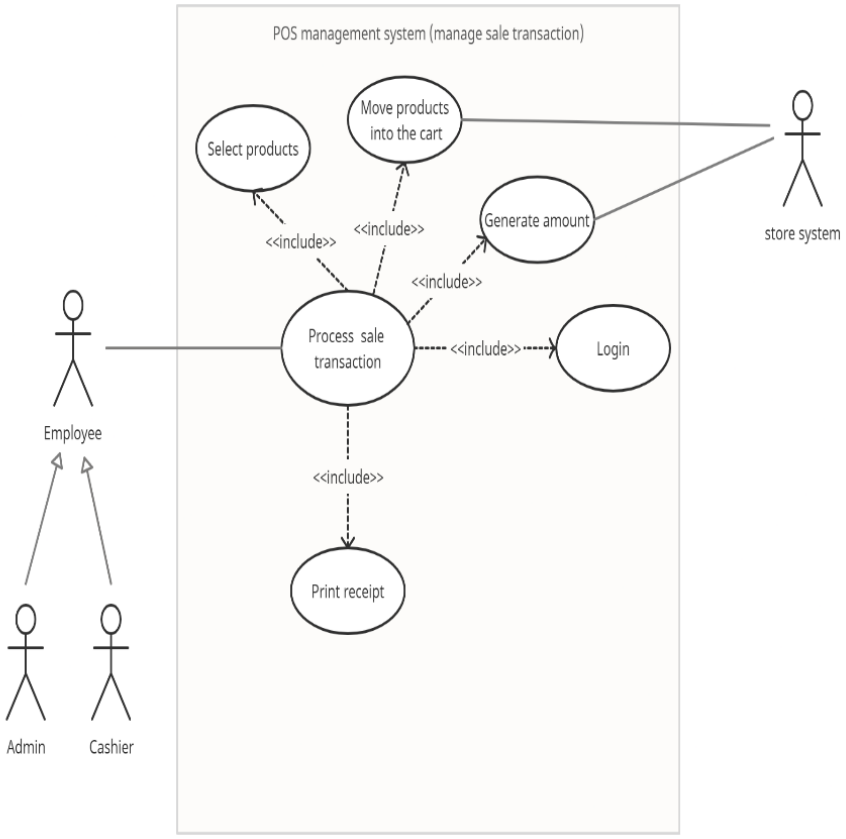
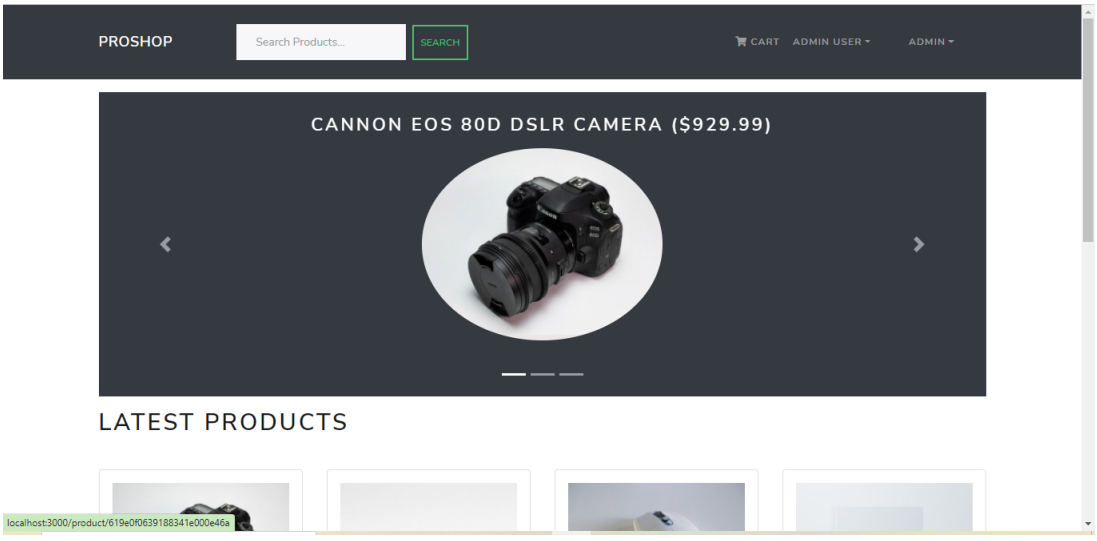


Figure 23: Transaction handling use case diagram

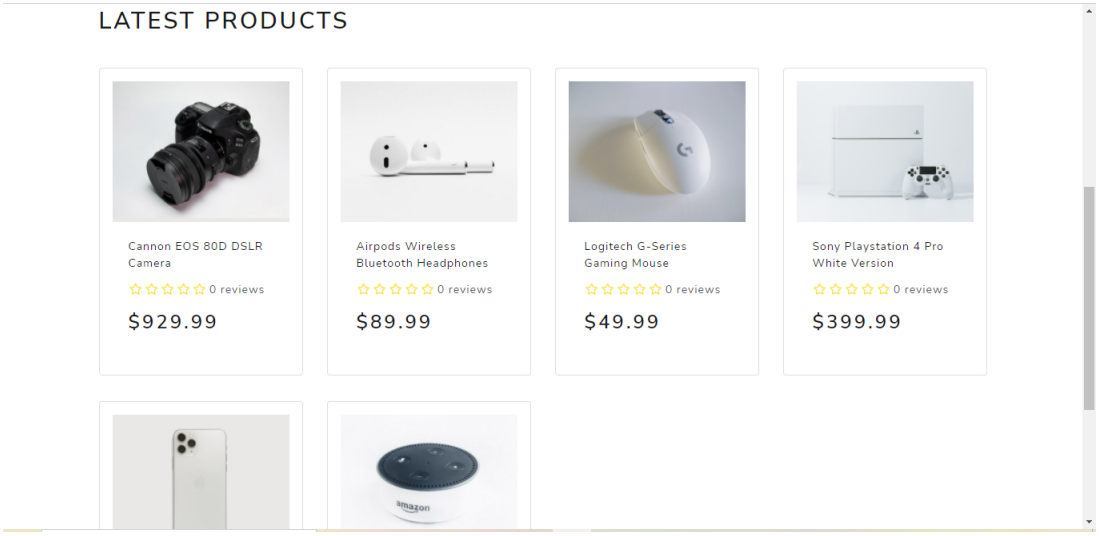
6.0 Design and Implementation of E-Commerce Site for Online Shopping

ONLINE SHOPPING APPLICATION: Anyone can view Online Shopping portal and available products, but every user must login by his/her Username and password in order to purchase or order products. Unregistered members can register by navigating to registration page. Only Admin will have access to modify roles, by default developer can only be an 'Admin'. Once user register site, his default role will be 'User'.

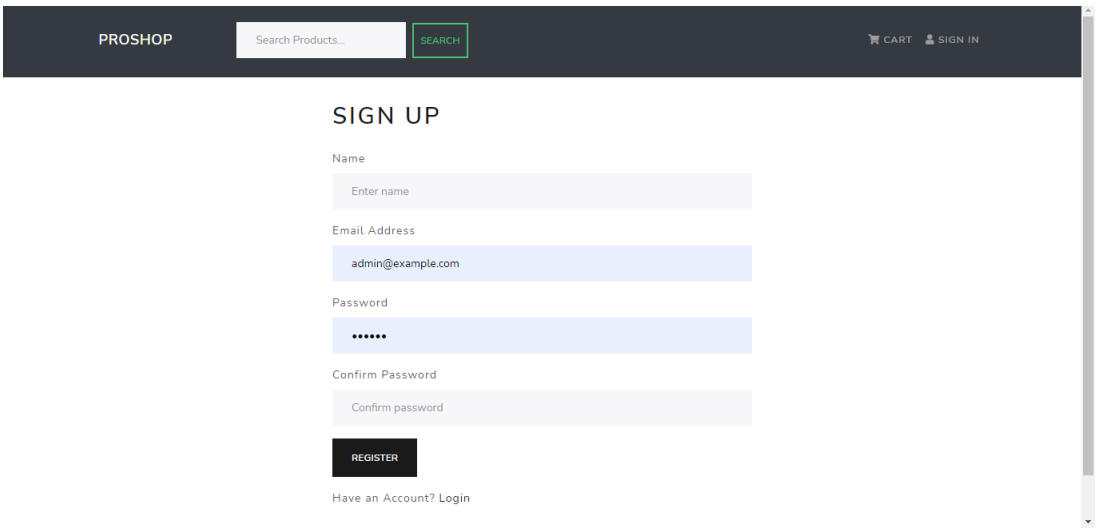
HOMEPAGE: The Home Screen will consist of screen were one can browse through the products which we have on our website Figure1: Home Page Design and Implementation of E-Commerce Site for Online Shopping



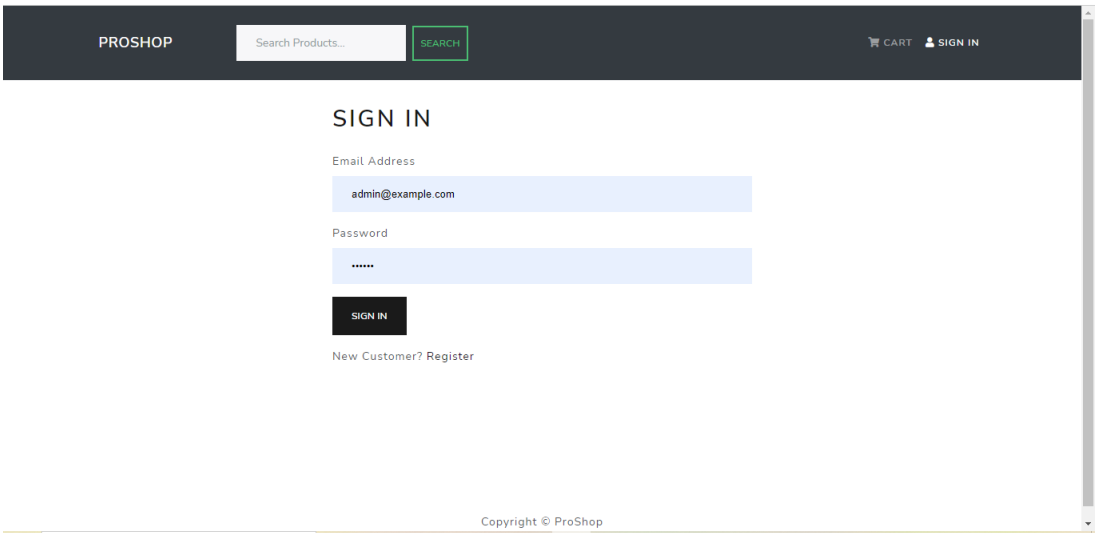
PRODUCTS PAGE: This page consists of product details. This page appears same for both visitors and users. Figure 2: Clothing Page 4.3 Order Us Page: Registered users can order desired products from here. Order Us Page Design and Implementation of E-Commerce Site for Online Shopping



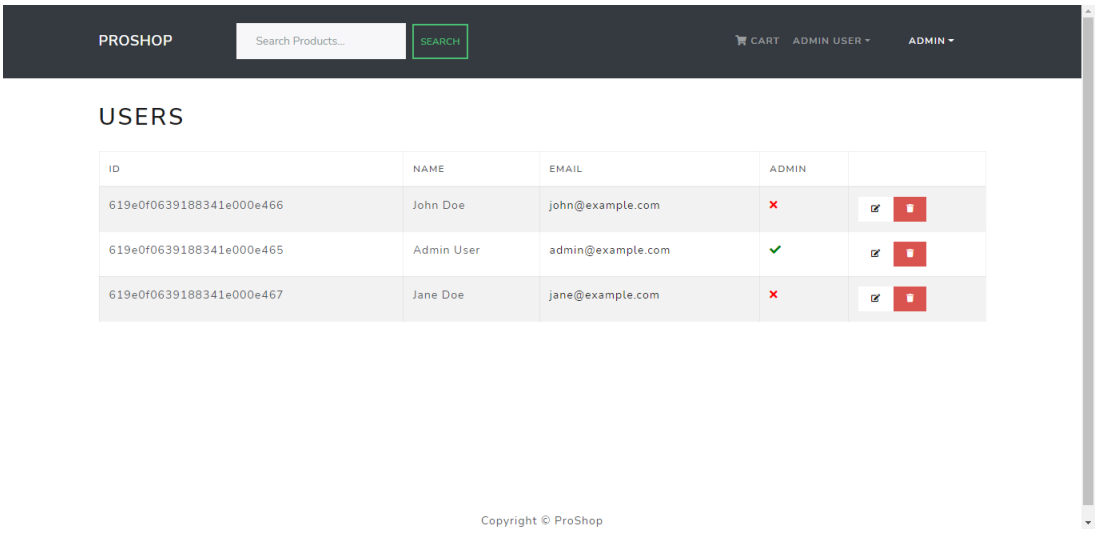
REGISTER PAGE: New users can register here Figure 7: Register Page Design and Implementation of E-Commerce Site for Online Shopping



LOGIN PAGE: Login page for both users and administrators.

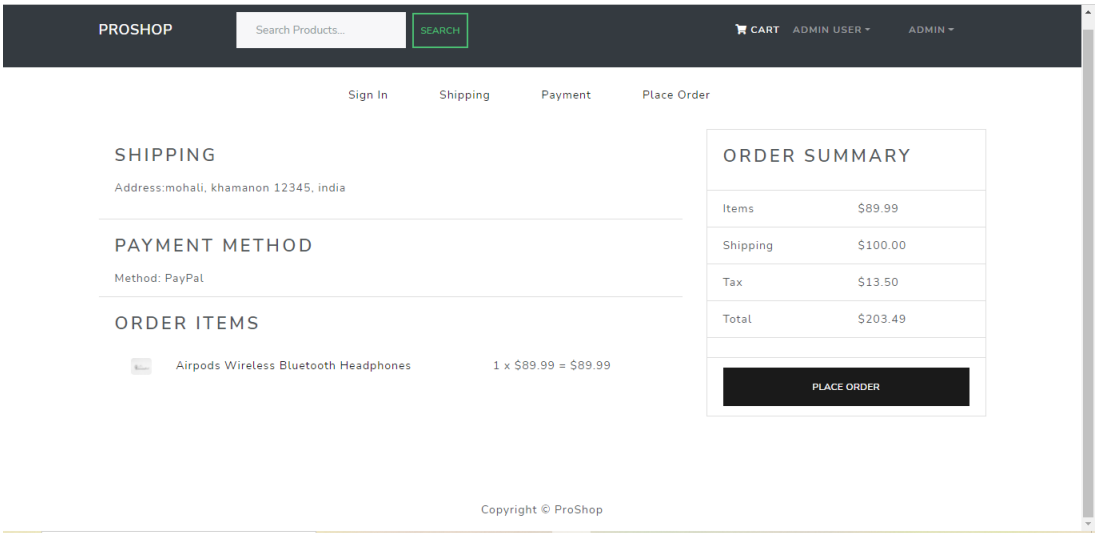


Admin Page: Only difference you see in this page is Role: Admin. User and Admin role will be checked once the page was login and Session ["role"] will be either Admin or User. If credentials belong to Admin then role will be Admin and if credentials belong to User then role will be User. Admin Page Design and Implementation of E-Commerce Site for Online Shopping

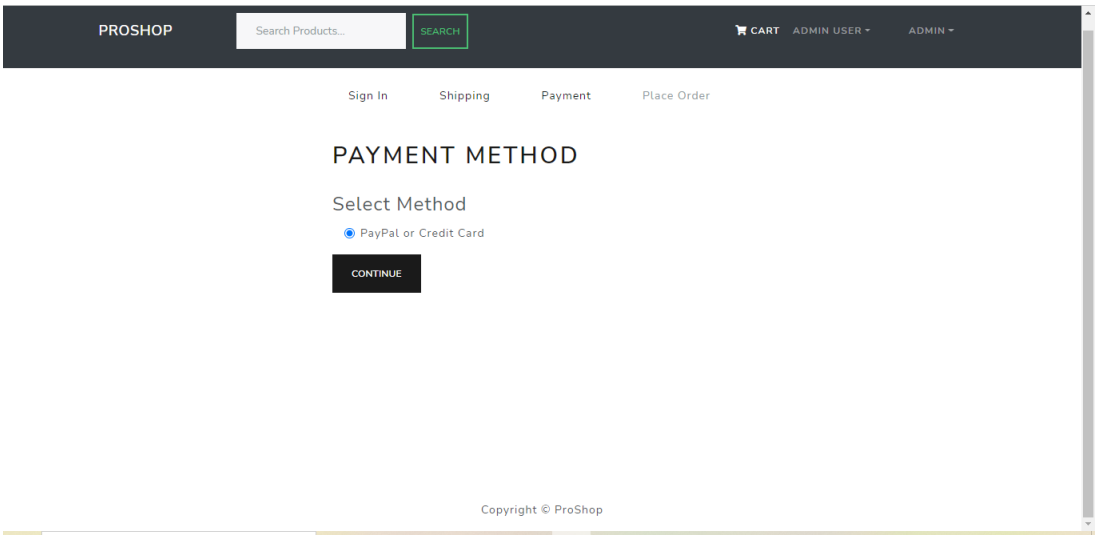


ORDER VIEW FOR USER: Once users order item they are able to see ordered products and grand total.

Order View for User



PAYPAL FOR PAYMENT: Once users orders products they are redirected to payment



7. CODING

order.controller.js

```
import asyncHandler from 'express-async-handler'
import Order from '../models/orderModel.js'

// @desc    Create new order
// @route   POST /api/orders
// @access  Private
const addOrderItems = asyncHandler(async (req, res) => {
const {
orderItems,
shippingAddress,
paymentMethod,
itemsPrice,
taxPrice,
shippingPrice,
```

```
totalPrice,
} = req.body

if (orderItems && orderItems.length === 0) {
res.status(400)
throw new Error('No order items')
return
} else {
const order = new Order({
orderItems,
user: req.user._id,
shippingAddress,
paymentMethod,
itemsPrice,
taxPrice,
shippingPrice,
totalPrice,
})

const createdOrder = await order.save()

res.status(201).json(createdOrder)
}
})

// @desc    Get order by ID
// @route   GET /api/orders/:id
// @access  Private
const getOrderById = asyncHandler(async (req, res) => {
const order = await Order.findById(req.params.id).populate(
```

```
'user',
'name email'
)

if (order) {
res.json(order)
} else {
res.status(404)
throw new Error('Order not found')
}
})

// @desc    Update order to paid
// @route   GET /api/orders/:id/pay
// @access  Private
const updateOrderToPaid = asyncHandler(async (req, res) => {
const order = await Order.findById(req.params.id)

if (order) {
order.isPaid = true
order.paidAt = Date.now()
order.paymentResult = {
id: req.body.id,
status: req.body.status,
update_time: req.body.update_time,
email_address: req.body.payer.email_address,
}

const updatedOrder = await order.save()
```

```
res.json(updatedOrder)
} else {
res.status(404)
throw new Error('Order not found')
}
})

// @desc    Update order to delivered
// @route   GET /api/orders/:id/deliver
// @access  Private/Admin
const updateOrderToDelivered = asyncHandler(async (req, res) => {
const order = await Order.findById(req.params.id)

if (order) {
order.isDelivered = true
order.deliveredAt = Date.now()

const updatedOrder = await order.save()

res.json(updatedOrder)
} else {
res.status(404)
throw new Error('Order not found')
}
})

// @desc    Get logged in user orders
// @route   GET /api/orders/myorders
// @access  Private
const getMyOrders = asyncHandler(async (req, res) => {
```

```
const orders = await Order.find({ user: req.user._id })
res.json(orders)
})

// @desc    Get all orders
// @route   GET /api/orders
// @access  Private/Admin
const getOrders = asyncHandler(async (req, res) => {
const orders = await Order.find({}).populate('user', 'id name')
res.json(orders)
})

export {
addOrderItems,
getOrderById,
updateOrderToPaid,
updateOrderToDelivered,
getMyOrders,
getOrders,
}
```

product.controller.js

```
import asyncHandler from 'express-async-handler'
import Order from '../models/orderModel.js'

// @desc    Create new order
// @route   POST /api/orders
// @access  Private
```

```
const addOrderItems = asyncHandler(async (req, res) => {
const {
orderItems,
shippingAddress,
paymentMethod,
itemsPrice,
taxPrice,
shippingPrice,
totalPrice,
} = req.body

if (orderItems && orderItems.length === 0) {
res.status(400)
throw new Error('No order items')
return
} else {
const order = new Order({
orderItems,
user: req.user._id,
shippingAddress,
paymentMethod,
itemsPrice,
taxPrice,
shippingPrice,
totalPrice,
})

const createdOrder = await order.save()

res.status(201).json(createdOrder)
```

```
}
})

// @desc    Get order by ID
// @route   GET /api/orders/:id
// @access  Private
const getOrderById = asyncHandler(async (req, res) => {
const order = await Order.findById(req.params.id).populate(
'user',
'name email'
)

if (order) {
res.json(order)
} else {
res.status(404)
throw new Error('Order not found')
}
})

// @desc    Update order to paid
// @route   GET /api/orders/:id/pay
// @access  Private
const updateOrderToPaid = asyncHandler(async (req, res) => {
const order = await Order.findById(req.params.id)

if (order) {
order.isPaid = true
order.paidAt = Date.now()
order.paymentResult = {
```

```
id: req.body.id,
status: req.body.status,
update_time: req.body.update_time,
email_address: req.body.payer.email_address,
}

const updatedOrder = await order.save()

res.json(updatedOrder)
} else {
res.status(404)
throw new Error('Order not found')
}
})

// @desc    Update order to delivered
// @route   GET /api/orders/:id/deliver
// @access  Private/Admin
const updateOrderToDelivered = asyncHandler(async (req, res) => {
const order = await Order.findById(req.params.id)

if (order) {
order.isDelivered = true
order.deliveredAt = Date.now()

const updatedOrder = await order.save()

res.json(updatedOrder)
} else {
res.status(404)
```

```
throw new Error('Order not found')
}
})

// @desc    Get logged in user orders
// @route   GET /api/orders/myorders
// @access  Private
const getMyOrders = asyncHandler(async (req, res) => {
const orders = await Order.find({ user: req.user._id })
res.json(orders)
})

// @desc    Get all orders
// @route   GET /api/orders
// @access  Private/Admin
const getOrders = asyncHandler(async (req, res) => {
const orders = await Order.find({}).populate('user', 'id name')
res.json(orders)
})

export {
addOrderItems,
getOrderById,
updateOrderToPaid,
updateOrderToDelivered,
getMyOrders,
getOrders,
}
```

user.controller.js

```
import asyncHandler from 'express-async-handler'
import generateToken from '../utils/generateToken.js'
import User from '../models/userModel.js'

// @desc    Auth user & get token
// @route   POST /api/users/login
// @access  Public
const authUser = asyncHandler(async (req, res) => {
const { email, password } = req.body

const user = await User.findOne({ email })

if (user && (await user.matchPassword(password))) {
res.json({
_id: user._id,
name: user.name,
email: user.email,
isAdmin: user.isAdmin,
token: generateToken(user._id),
})
} else {
res.status(401)
throw new Error('Invalid email or password')
}
})

// @desc    Register a new user
// @route   POST /api/users
```

```
// @access  Public
const registerUser = asyncHandler(async (req, res) => {
const { name, email, password } = req.body

const userExists = await User.findOne({ email })

if (userExists) {
res.status(400)
throw new Error('User already exists')
}

const user = await User.create({
name,
email,
password,
})

if (user) {
res.status(201).json({
_id: user._id,
name: user.name,
email: user.email,
isAdmin: user.isAdmin,
token: generateToken(user._id),
})
} else {
res.status(400)
throw new Error('Invalid user data')
}
})
```

```
// @desc    Get user profile
// @route   GET /api/users/profile
// @access  Private
const getUserProfile = asyncHandler(async (req, res) => {
const user = await User.findById(req.user._id)

if (user) {
res.json({
_id: user._id,
name: user.name,
email: user.email,
isAdmin: user.isAdmin,
})
} else {
res.status(404)
throw new Error('User not found')
}
})

// @desc    Update user profile
// @route   PUT /api/users/profile
// @access  Private
const updateUserProfile = asyncHandler(async (req, res) => {
const user = await User.findById(req.user._id)

if (user) {
user.name = req.body.name || user.name
user.email = req.body.email || user.email
if (req.body.password) {
```

```
user.password = req.body.password
}

const updatedUser = await user.save()

res.json({
_id: updatedUser._id,
name: updatedUser.name,
email: updatedUser.email,
isAdmin: updatedUser.isAdmin,
token: generateToken(updatedUser._id),
})
} else {
res.status(404)
throw new Error('User not found')
}
})

// @desc    Get all users
// @route   GET /api/users
// @access  Private/Admin
const getUsers = asyncHandler(async (req, res) => {
const users = await User.find({})
res.json(users)
})

// @desc    Delete user
// @route   DELETE /api/users/:id
// @access  Private/Admin
const deleteUser = asyncHandler(async (req, res) => {
```

```
const user = await User.findById(req.params.id)


if (user) {
await user.remove()
res.json({ message: 'User removed' })
} else {
res.status(404)
throw new Error('User not found')
}
})

// @desc    Get user by ID
// @route   GET /api/users/:id
// @access  Private/Admin
const getUserById = asyncHandler(async (req, res) => {
const user = await User.findById(req.params.id).select('-password')


if (user) {
res.json(user)
} else {
res.status(404)
throw new Error('User not found')
}
})


// @desc    Update user
// @route   PUT /api/users/:id
// @access  Private/Admin
const updateUser = asyncHandler(async (req, res) => {
const user = await User.findById(req.params.id)
```

```
if (user) {
user.name = req.body.name || user.name
user.email = req.body.email || user.email
user.isAdmin = req.body.isAdmin

const updatedUser = await user.save()

res.json({
_id: updatedUser._id,
name: updatedUser.name,
email: updatedUser.email,
isAdmin: updatedUser.isAdmin,
})
} else {
res.status(404)
throw new Error('User not found')
}
})

export {
authUser,
registerUser,
getUserProfile,
updateUserProfile,
getUsers,
deleteUser,
getUserById,
updateUser,
}
```

order.model.js

```
import mongoose from 'mongoose'


const orderSchema = mongoose.Schema(
{
user: {
type: mongoose.Schema.Types.ObjectId,
required: true,
ref: 'User',
},
orderItems: [
{
name: { type: String, required: true },
qty: { type: Number, required: true },
image: { type: String, required: true },
price: { type: Number, required: true },
product: {
type: mongoose.Schema.Types.ObjectId,
required: true,
ref: 'Product',
},
},
],
shippingAddress: {
address: { type: String, required: true },
```

```
city: { type: String, required: true },
postalCode: { type: String, required: true },
country: { type: String, required: true },
},
paymentMethod: {
type: String,
required: true,
},
paymentResult: {
id: { type: String },
status: { type: String },
update_time: { type: String },
email_address: { type: String },
},
taxPrice: {
type: Number,
required: true,
default: 0.0,
},
shippingPrice: {
type: Number,
required: true,
default: 0.0,
},
totalPrice: {
type: Number,
required: true,
default: 0.0,
},
isPaid: {
```

```
type: Boolean,
required: true,
default: false,
},
paidAt: {
type: Date,
},
isDelivered: {
type: Boolean,
required: true,
default: false,
},
deliveredAt: {
type: Date,
},
},
{
timestamps: true,
}
)

const Order = mongoose.model('Order', orderSchema)

export default Order
```

product.model.js

```
import mongoose from "mongoose";


const reviewSchema = mongoose.Schema(
{
name: { type: String, required: true },
rating: { type: Number, required: true },
comment: { type: String, required: true },
user: {
type: mongoose.Schema.Types.ObjectId,
required: true,
ref: "User",
},
},
{
timestamps: true,
}
);


const productSchema = mongoose.Schema(
{
user: {
type: mongoose.Schema.Types.ObjectId,
// required: true,
ref: "User",
},
name: {
type: String,
required: true,
},
image: {
```

```
type: String,
required: true,
},
brand: {
type: String,
required: true,
},
category: {
type: String,
// required: true,
},
description: {
type: String,
required: true,
},
reviews: [reviewSchema],
rating: {
type: Number,
required: true,
default: 0,
},
numReviews: {
type: Number,
// required: true,
default: 0,
},
price: {
type: Number,
required: true,
default: 0,
```

```
},
countInStock: {
type: Number,
required: true,
default: 0,
},
},
{
timestamps: true,
}
);

const Product = mongoose.model("Products", productSchema);

export default Product;
```

user.model.js

```
import mongoose from 'mongoose'
import bcrypt from 'bcryptjs'

const userSchema = mongoose.Schema(
{
name: {
type: String,
required: true,
},
email: {
type: String,
```

```
required: true,
unique: true,
},
password: {
type: String,
required: true,
},
isAdmin: {
type: Boolean,
required: true,
default: false,
},
},
{
timestamps: true,
}
)

userSchema.methods.matchPassword = async function (enteredPassword) {
return await bcrypt.compare(enteredPassword, this.password)
}

userSchema.pre('save', async function (next) {
if (!this.isModified('password')) {
next()
}

const salt = await bcrypt.genSalt(10)
this.password = await bcrypt.hash(this.password, salt)
})
```

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

export default User
```

Authmiddleware.js

```
import jwt from 'jsonwebtoken'
import asyncHandler from 'express-async-handler'
import User from '../models/userModel.js'

const protect = asyncHandler(async (req, res, next) => {
let token

if (
req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')
) {
try {
token = req.headers.authorization.split(' ')[1]

const decoded = jwt.verify(token, process.env.JWT_SECRET)

req.user = await User.findById(decoded.id).select('-password')

next()
} catch (error) {
```

```
console.error(error)
res.status(401)
throw new Error('Not authorized, token failed')
}
}

if (!token) {
res.status(401)
throw new Error('Not authorized, no token')
}
})

const admin = (req, res, next) => {
if (req.user && req.user.isAdmin) {
next()
} else {
res.status(401)
throw new Error('Not authorized as an admin')
}
}

export { protect, admin }
```

errorMidlleware.js

```
const notFound = (req, res, next) => {
const error = new Error(`Not Found - ${req.originalUrl}`)
res.status(404)
next(error)
```

```
}

const errorHandler = (err, req, res, next) => {
const statusCode = res.statusCode === 200 ? 500 : res.statusCode
res.status(statusCode)
res.json({
message: err.message,
stack: process.env.NODE_ENV === 'production' ? null : err.stack,
})
}

export { notFound, errorHandler }
```

config db.js

```
import mongoose from "mongoose";

const connectDB = async () => {
try {
const conn = await mongoose.connect(process.env.MONGO_URI, {
useUnifiedTopology: true,
useNewUrlParser: true,
useCreateIndex: true,
});

console.log(`MongoDB Connected: ${conn.connection.host}`.cyan.underline);
} catch (error) {
console.error(`Error: ${error.message}`.red.underline.bold);
process.exit(1);
```

```
}
};

export default connectDB;
```

server.js

```
import path from "path";
import express from "express";
import dotenv from "dotenv";
import colors from "colors";
import morgan from "morgan";
import { notFound, errorHandler } from "./middleware/errorMiddleware.js";
import connectDB from "./config/db.js";

import productRoutes from "./routes/productRoutes.js";
import userRoutes from "./routes/userRoutes.js";
import orderRoutes from "./routes/orderRoutes.js";
import uploadRoutes from "./routes/uploadRoutes.js";

dotenv.config();

connectDB();

const app = express();

if (process.env.NODE_ENV === "development") {
app.use(morgan("dev"));
```

```
}

app.use(express.json());

app.use("/api/products", productRoutes);
app.use("/api/users", userRoutes);
app.use("/api/orders", orderRoutes);
app.use("/api/upload", uploadRoutes);

app.get("/api/config/paypal", (req, res) =>
res.send(process.env.PAYPAL_CLIENT_ID)
);

const __dirname = path.resolve();
app.use("/uploads", express.static(path.join(__dirname, "/uploads")));

if (process.env.NODE_ENV === "production") {
app.use(express.static(path.join(__dirname, "/frontend/build")));

app.get("*", (req, res) =>
res.sendFile(path.resolve(__dirname, "frontend", "build", "index.html"))
);
} else {
app.get("/", (req, res) => {
res.send("API is running....");
});
}

app.use(notFound);
app.use(errorHandler);
```

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


app.listen(
PORT,
console.log(
`Server running in ${process.env.NODE_ENV} mode on port ${PORT}`.yellow.bold
)
);
```

8. APPROACH USED

For designing the algorithm for developing this application, we used “Top-down approach”.

The top-down approach basically divides a complex problem or algorithm into multiple smaller parts (modules). These modules are further decomposed until the resulting module is the fundamental program essentially be understood and cannot be further decomposed. After achieving a certain level of modularity, the decomposition of modules is ceased. The top-down approach is the stepwise process of breaking of the large program module into simpler and smaller modules to organize and code program in an efficient way. The flow of control in this approach is always in the downward direction.

9. TESTING

Test Cases & Test Criteria

Test Cases:

Cashier activity

Test the entry of items purchased by a customer is correct

Check totals and closings match

Sales

Check for a regular sale process

Check for correct prices are displayed for merchandise purchased

Test for "0" or null transaction

Test for billing details or shipping details in payment manager

Test for reference transaction

Performance

Check for speed or time taken to receive a response or send a request

Security & Regulatory Compliance

Test user profiles and access levels

Test database consistency

Verify specific information about user, transaction, product and category

10. Implementation & Evaluation of Project

This is the most important part of for finalizing the software development. Before we hand over the project, we need to justify the system requirements so that the client can run the system

without any obligations and dissatisfactions. For that below is our requirements:

10.1. Technology used

To develop this project, MERN stack technology is used.

Introduction of MERN stack technologies

MERN is an acronym used to describe a specific set of JavaScript based technologies that are used in the web application development process. It is designed with an idea

to make the development process as smooth as possible. MERN includes the following open-source components:

- MongoDB: A document-oriented, No-SQL database used to store the application data.
- NodeJS: The JavaScript runtime environment. It is used to run JavaScript on a machine rather than in a browser.
- ExpressJS: A framework layered on top of NodeJS, used to build the backend of a site using NodeJS functions and structures. Since NodeJS was not developed to make websites but rather run JavaScript on a machine, ExpressJS was developed.
- ReactJS: A library created by Facebook. It is used to build UI components that create the user interface of the single page web application.

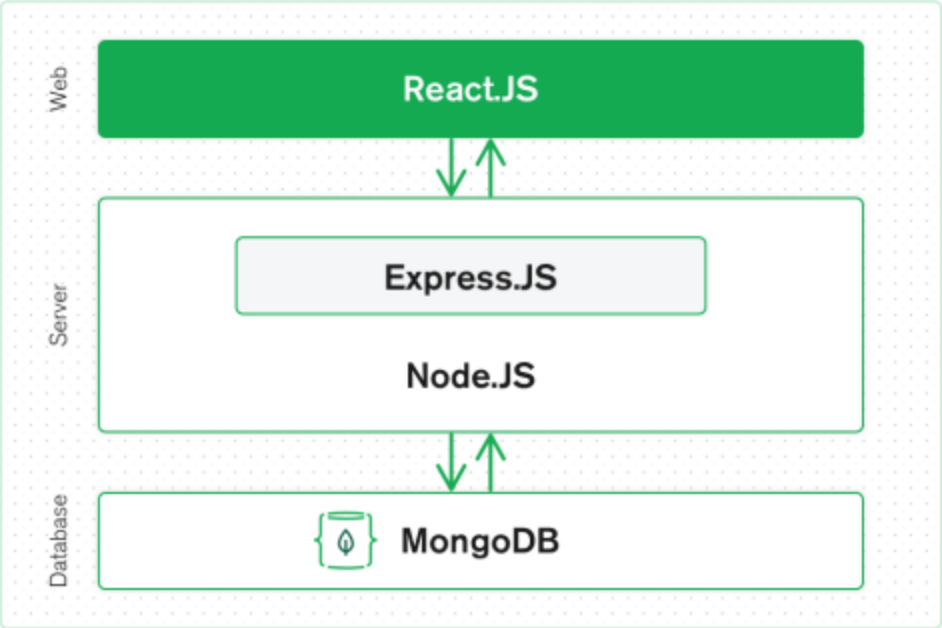


Figure 32: MERN technology structure

10.2. Software requirements

. Operating System: Windows (XP, 7, 8, 8.1, 10) or Mac OS

. Database Management System: MongoDB Compass

. IDE (Integrated Development Environment): Visual Studio code

10.3. Hardware requirements

RAM: Minimum 1GB or higher.

HDD: Minimum 50 GB.

Processor: Intel Pentium 4 or higher.

10.4. System Requirements

. Development

React JS version 17.0.0

NODE JS version 14.15.5

mongoDB version 4.4.0

Express js version 4.17

GIT version control

NPM (Node package Manager)

. Production

The Syncfusion Essential JS 2 components are supported only in modern browsers. This includes the following versions.

Chrome	Firefox	Opera	Edge	IE	Safari	IOS	Android	Windows Mobile
Latest	Latest	Latest	13 +	11 +	9 +	9 +	4.4 +	IE 11 +

11.0 SCOPE OF PROJECT

Before we launch into the future and scope of eCommerce in India, let us first understand what is e-commerce. To put it simply, electronic commerce refers to the purchase and sale of goods online or via the internet.

Sellers make websites where they display images of their products with price and description. Shoppers who buy the products have multiple payment options like COD, e-wallet, net banking, credit card, and so on.

Online sellers have the responsibility of shipping the product to the buyer and ensuring safe and timely delivery.

12.0 CONCLUSION

Technology has made significant progress over the years to provide consumers a better online shopping experience and will continue to do so for years to come. With the rapid growth of products and brands, people have speculated that online shopping will overtake in-store shopping. While this has been the case in some areas, there is still demand for brick and mortar stores in market areas where the consumer feels more comfortable seeing and touching the product being bought. However, the availability of online shopping has produced a more educated consumer that can shop around with relative ease without having to spend a large amount of time. In exchange, online shopping has opened up doors to many small retailers that would never be in business if they had to incur the high cost of owning a brick and mortar store. At the end, it has been a win-win situation for both consumer and sellers.

13.0 BIBLIOGRAPHY

- https://www.shareitsolutions.com/blog/mern-technology-stack
- https://en.itpedia.nl/2019/01/16/wat-heeft-een-functioneel-ontwerp-te-bieden/
- https://www.slideshare.net/sharathgrao/fod-24996475
- https://www.tutorialspoint.com/software_engineering/software_design_basics.htm
- https://economictimes.indiatimes.com/definition/systems-design
- https://en.wikipedia.org/wiki/
- https://belitsoft.com/custom-application-development-services/software-requirements-specification-document-example-international-standard/