Pune Institute of Computer Technology
Dhankawadi, Pune

A MINI PROJECT REPORT
ON

E-COMMERCE APPLICATION FOR GARMENTS

SUBMITTED BY

Student Name: Niraj Sarode	**Roll No:31456**
Student Name: Geet Shingi	**Roll No:31457**
Student Name: Utkarsh Bhure	**Roll No:31465**
Student Name: Vedangi Wagh	**Roll No:31468**

Under the guidance of
Prof. A. A. Jewalikar

DEPARTMENT OF COMPUTER ENGINEERING
Academic Year 2020-21

DEPARTMENT OF COMPUTER ENGINEERING

Pune Institute of Computer Technology

Dhankawadi, Pune-43

CERTIFICATE

This is to certify that the Mini-Project report entitled

"E-COMMERCE APPLICATION FOR GARMENTS"

Submitted by

Student Name: Niraj Sarode	**Roll No:31456**
Student Name: Geet Shingi	**Roll No:31457**
Student Name: Utkarsh Bhure	**Roll No:31465**
Student Name: Vedangi Wagh	**Roll No:31468**

has satisfactorily completed a mini-project report under the guidance of Prof. A. A. Jewalikar towards the partial fulfillment of third year Computer Engineering Semester II, Academic Year 2019-20 of Savitribai Phule Pune University.

Prof. A. A. Jewalikar
Internal Guide

Prof. M.S.Takalikar
Head
Department of Computer Engineering

Place:Pune
Date:

ACKNOWLEDGEMENT

I sincerely thank our WTL Coordinator Prof. A. A. Jewalikar and Head of Department Prof. M.S.Takalikar for their support.

I also sincerely convey my gratitude to my guide Prof. A. A. Jewalikar, Department of Computer Engineering for her constant support, providing all the help, motivation and encouragement from beginning till end.

Contents

List of Figures

1 PROJECT IDEA AND FUNCTIONAL REQUIREMENTS

1.1 Project Idea

In today's fast-changing business environment, it's extremely important to be able to respond to client needs in the most effective and timely manner. If your customers wish to see your business online and have instant access to your products or services then an E-Commerce Application is required.

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.

With the booming of e-commerce market in the 21st century, more and more consumers are changing their shopping behavior and prefer to do their usual shopping from the comfort of their homes rather than putting up with the time consuming physical store deal. Keeping this in mind 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. 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.

1.2 Functional Requirements

1. USER

   - USER LOGIN
     Description of feature:
     This feature used by the user to login into system. A user must login with his user name and password to the system after registration. If they are invalid, the user not allowed to enter the system.
     Functional requirement:
     -Username and password will be provided after user registration is confirmed.
     -Password should be hidden from others while typing it in the field

- REGISTER NEW USER
  Description of feature:
  A new user will have to register in the system by providing essential details in order to view the products in the system.
  Functional requirement:
  -System must be able to verify and validate information.
  -The system must encrypt the password of the customer to provide security.
- PURCHASING AN ITEM
  Description of feature:
  The user can add the desired product into his cart by clicking add to cart option on the product. He can view his cart by clicking on the cart button. All products added by cart can be viewed in the cart. User can remove an item from the cart by clicking remove.After confirming the items in the cart the user can submit the cart by providing a delivery address.On successful submitting the cart will become empty.
  Functional requirement:
  -System must ensure that, only a registered customer can purchase items.

2. ADMIN

- MANAGE USER
  Description of feature:
  The administrator can add user, delete user, view user and block user.
  Functional requirement:
  -Must fulfill basic CRUD operations related to users.
- MANAGE PRODUCTS
  Description of feature:
  The administrator can add product, delete product and view product.
  Functional requirement:
  -Must fulfill basic CRUD operations related to products.
- MANAGE ORDERS
  Description of feature The administrator can view orders and delete orders.
  Functional requirements:
  -The system must identify the login of the admin.
  -Admin account should be secured so that only owner of the shop can access that account.

2 DESIGN

2.1 Use Case Diagram

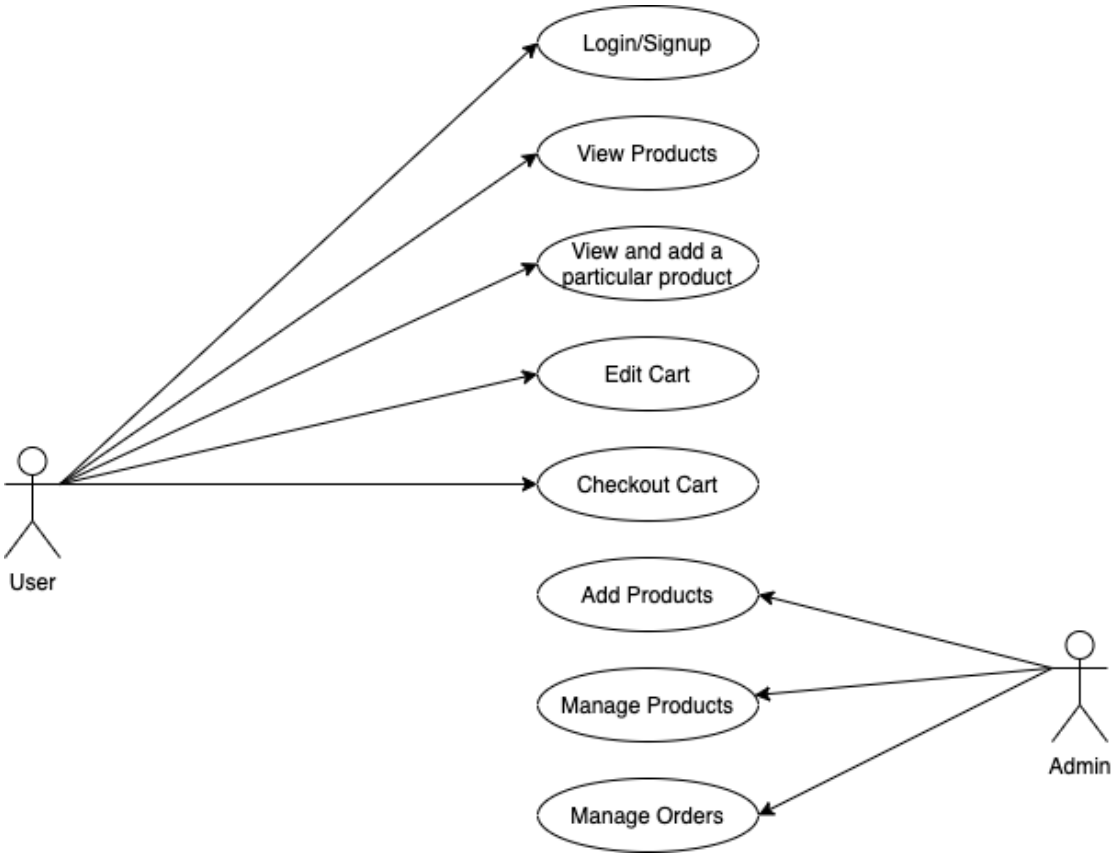

Figure 1: Application Use Case

2.2 Database Schema

User Schema

```
var User = new Schema({
    firstname: {
      type: String,
        default: ''
    },
    lastname: {
      type: String,
        default: ''
    },
    admin: {
        type: Boolean,
        default: false
    },
});
```

Product Schema

```
var commentSchema = new Schema({
    rating: {
        type: Number,
        min: 1,
        max: 5,
        required: true
    },
    comment: {
        type: String,
        required: true
    },
    author: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    }
}, {
    timestamps: true
});
var sizeSchema = new Schema({
    size: {
    type: String,
    required: true
    },
    quantity: {
    type: Number,
    min: 0,
    required: true
    }
});
const productSchema = new Schema({
    name: {
        type: String,
        required: true,
        unique: true
    },
    description: {
        type: String,
        required: true
    },
     price: {
        type: String,
        required: true,
        min: 0
    },
     image: {
         type: [String],
         required: true
     },
    size:[sizeSchema],
```

```
    comments:[commentSchema]
}, {
    timestamps: true
});
```

Cart and Order Schema

```
var cartSchema = new Schema({
    user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    },
    products: [{
        name: {
         type: String,
         required: true
        },
        description: {
         type: String,
         required: true
        },
        size: {
         type: String,
         required: true
        },
        quantity: {
         type: Number,
         required: true
        },
        price: {
         type: Number,
         required: true
        }
    }],
    delivered: {
        type: Boolean,
        default: false
    },
}, {
    timestamps: true
});
```

3 SOURCE CODE AND SCREENSHOTS

3.1 NodeJS Code

3.1.1 App.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var session = require('express-session');
var FileStore = require('session-file-store')(session);
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var config = require('./config.js');
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var productRouter = require('./routes/productRouter');
var cartRouter = require('./routes/cartRouter');
var uploadRouter = require('./routes/uploadRouter');
var orderRouter = require('./routes/orderRouter');
var app = express();
var passport = require('passport');
var authenticate = require('./authenticate');
const mongoose = require('mongoose');
const url = config.mongoUrl;
const connect = mongoose.connect(url);

connect.then((db) => {
    console.log("Connected correctly to server");
}, (err) => { console.log(err); });
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(passport.initialize());
app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use(express.static(path.join(__dirname, 'public')));
app.use('/imageUpload', uploadRouter);
app.use('/products', productRouter);
app.use('/cart', cartRouter);
app.use('/orders', orderRouter);
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
```

```
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

3.1.2 productRouter.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
var authenticate = require('../authenticate');
const cors = require('./cors');

const Products = require('../models/product');

const productRouter = express.Router();

productRouter.use(bodyParser.json());

productRouter.route('/')
.options(cors.corsWithOptions, (req, res) => { res.sendStatus(200); })
.get(cors.cors, (req,res,next) => {
    Products.find(req.query)
    .then((products) => {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(products);
    }, (err) => next(err))
    .catch((err) => next(err));
})
.post(cors.corsWithOptions, authenticate.verifyUser,
    authenticate.verifyAdmin, (req, res, next) => {
    Products.create(req.body)
    .then((product) => {
        console.log('Product Created ', product);
        res.statusCode = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(product);
    }, (err) => next(err))
    .catch((err) => next(err));
})
.put(cors.corsWithOptions, authenticate.verifyUser, (req, res, next) =>
    {
```

```
    res.statusCode = 403;
    res.end('PUT operation not supported on /products');
})
.delete(cors.corsWithOptions, authenticate.verifyUser,
    authenticate.verifyAdmin, (req, res, next) => {
    Products.remove({})
    .then((resp) => {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(resp);
    }, (err) => next(err))
    .catch((err) => next(err));
});

productRouter.route('/:productId')
.options(cors.corsWithOptions, (req, res) => { res.sendStatus(200); })
.get(cors.cors,(req,res,next) => {
    Products.findById(req.params.productId)
    .then((product) => {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(product);
    }, (err) => next(err))
    .catch((err) => next(err));
})
.post(cors.corsWithOptions, authenticate.verifyUser, (req, res, next)
    => {
    res.statusCode = 403;
    res.end('POST operation not supported on /products/'+
        req.params.productId);
})
.put(cors.corsWithOptions, authenticate.verifyUser, (req, res, next) =>
    {
    Products.findByIdAndUpdate(req.params.productId, {
        $set: req.body
    }, { new: true })
    .then((product) => {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(product);
    }, (err) => next(err))
    .catch((err) => next(err));
})
.delete(cors.corsWithOptions, authenticate.verifyUser, (req, res, next)
    => {
    Products.findByIdAndRemove(req.params.productId)
    .then((resp) => {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(resp);
    }, (err) => next(err))
```

```
    .catch((err) => next(err));
});

productRouter.route('/:productId/comments')
.options(cors.corsWithOptions, (req, res) => { res.sendStatus(200); })
.get(cors.cors, (req,res,next) => {
    Products.findById(req.params.productId)
    .populate('comments.author')
    .then((product) => {
        if (product != null) {
            res.statusCode = 200;
            res.setHeader('Content-Type', 'application/json');
            res.json(product.comments);
        }
        else {
            err = new Error('Product ' + req.params.productId + ' not
                found');
            err.status = 404;
            return next(err);
        }
    }, (err) => next(err))
    .catch((err) => next(err));
})
.post(cors.corsWithOptions, authenticate.verifyUser, (req, res, next)
    => {
    Products.findById(req.params.productId)
    .then((product) => {
        if (product != null) {
            req.body.author = req.user._id;
            product.comments.push(req.body);
            product.save()
            .then((product) => {
                Products.findById(product._id)
                .populate('comments.author')
                .then((product) => {
                    res.statusCode = 200;
                    res.setHeader('Content-Type', 'application/json');
                    res.json(product);
                })
            }, (err) => next(err));
        }
        else {
            err = new Error('Product ' + req.params.dishId + ' not
                found');
            err.status = 404;
            return next(err);
        }
    }, (err) => next(err))
    .catch((err) => next(err));
})
```

```
.put(cors.corsWithOptions, authenticate.verifyUser, (req, res, next) =>
    {
    res.statusCode = 403;
    res.end('PUT operation not supported on /products/'
        + req.params.productId + '/comments');
})
.delete(cors.corsWithOptions, authenticate.verifyUser,
    authenticate.verifyAdmin, (req, res, next) => {
    Products.findById(req.params.productId)
    .then((product) => {
        if (product != null) {
            for (var i = (product.comments.length -1); i >= 0; i--) {
                product.comments.id(product.comments[i]._id).remove();
            }
            product.save()
            .then((product) => {
                res.statusCode = 200;
                res.setHeader('Content-Type', 'application/json');
                res.json(product);
            }, (err) => next(err));
        }
        else {
            err = new Error('Product ' + req.params.productId + ' not
                found');
            err.status = 404;
            return next(err);
        }
    }, (err) => next(err))
    .catch((err) => next(err));
});
module.exports = productRouter;
```

3.1.3 uploadRouter.js

```
const express = require('express');
const bodyParser = require('body-parser');
const authenticate = require('../authenticate');
const multer = require('multer');
const cors = require('./cors');
var storage = multer.diskStorage({
  // destination
  destination: function (req, file, cb) {
    cb(null, './public/images/')
  },
  filename: function (req, file, cb) {
    cb(null, file.originalname);
  }
});

const imageFileFilter = (req, file, cb) => {
```

```
    if(!file.originalname.match(/\.(jpg|JPG|jpeg|JPEG|png|PNG|gif|GIF)$/))
        {
        return cb(new Error('You can upload only image files!'), false);
    }
    cb(null, true);
};

var upload = multer({ storage: storage, fileFilter: imageFileFilter});

const uploadRouter = express.Router();

uploadRouter.use(bodyParser.json());

uploadRouter.route('/')
.options(cors.corsWithOptions, (req, res) => { res.sendStatus(200); })
.get(authenticate.verifyUser, authenticate.verifyAdmin, (req, res,
    next) => {
    res.statusCode = 403;
    res.end('GET operation not supported on /imageUpload');
})
.post(cors.corsWithOptions, authenticate.verifyUser,
    authenticate.verifyAdmin, upload.array("uploads[]", 3), (req, res)
    => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'application/json');
    res.json(req.file);
})
.put(cors.corsWithOptions, authenticate.verifyUser,
    authenticate.verifyAdmin, (req, res, next) => {
    res.statusCode = 403;
    res.end('PUT operation not supported on /imageUpload');
})
.delete(cors.corsWithOptions, authenticate.verifyUser,
    authenticate.verifyAdmin, (req, res, next) => {
    res.statusCode = 403;
    res.end('DELETE operation not supported on /imageUpload');
});

module.exports = uploadRouter;
```

3.1.4 orderRouter.js

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const Order = require('../models/order');
const authenticate = require('../authenticate');
const cors = require('./cors');
const nodeCCAvenue = require('node-ccavenue');
const ccav = new nodeCCAvenue.Configure({
```

```
    merchant_id: process.env.test_merchant_id ||
        process.env.prod_merchant_id,
    working_key: process.env.test_working_key ||
        process.env.prod_working_key
  });

const orderRouter = express.Router();
orderRouter.use(bodyParser.json());
orderRouter.route('/checkout')
    .options(cors.corsWithOptions, (req, res) => { res.sendStatus(200);
        })
    .post(cors.corsWithOptions, authenticate.verifyUser, (req, res,
        next) => {

  const { encResp } = req.body;
  const output = ccav.redirectResponseToJson(encResp);

  logger.log(output);
  // The 'output' variable is the CCAvenue Response in JSON Format

  if(output.order_status === 'Failure') {
     // DO YOUR STUFF
    res.writeHead(301,
      {Location: 'http://localhost:4200/cart'}
    );
    res.end();
  } else if (output.order_status === 'Success') {
      // DO YOUR STUFF
      res.writeHead(301,
        {Location: 'http://localhost:4200/'}
      );
      res.end();
    }
});
orderRouter.route('/admin')
    .options(cors.corsWithOptions, (req, res) => { res.sendStatus(200);
        })
    .get(cors.corsWithOptions, authenticate.verifyUser,
        authenticate.verifyAdmin, (req, res, next) => {
        Order.find(req.query).populate('user').then((orders) => {
            res.status = 200;
            res.setHeader('Content-Type', 'application/json');
            res.json(orders);
        }, (err) => next(err)
        ).catch((err) => next(err));
    })
orderRouter.route('/')
    .options(cors.corsWithOptions, (req, res) => { res.sendStatus(200);
        })
    .get(cors.corsWithOptions, authenticate.verifyUser, (req, res,
        next) => {
```

```
    Order.findOne({user:
        req.user._id}).populate('user').then((orders) => {
        res.status = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(orders);
    }, (err) => next(err)
    ).catch((err) => next(err));
})
.post(cors.corsWithOptions, authenticate.verifyUser, (req, res,
    next) => {
    Order.findOne({user: req.user._id}).then((order) => {
        if (!order) {
            Order.create({"user": req.user._id, "orders":
                req.body}).then((order) => {
                res.status = 200;
                res.setHeader('Content-Type', 'application/json');
                res.json(order);
            });
        }
        else {
                order.orders.push(req.body);
                order.save().then((orders) => {
                res.status = 200;
                res.setHeader('Content-Type', 'application/json');
                res.json(orders);
            }, (err) => next(err))
            .catch((err) => next(err));
        }
    }, (err) => next(err))
        .catch((err) => next(err));
})
.put(cors.corsWithOptions, authenticate.verifyUser, (req, res,
    next) => {
    res.statusCode = 403;
    res.end('PUT operation not supported on /orders');
})
.delete(cors.corsWithOptions, authenticate.verifyUser, (req, res,
    next) => {
    Order.findOneAndDelete({user: req.user._id}).then((resp) => {
        res.status = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(resp);
    }, (err) => next(err))
    .catch((err) => next(err));
});

orderRouter.route('/:cartId')
.options(cors.corsWithOptions, (req, res) => { res.sendStatus(200);
    })
.get(cors.cors, authenticate.verifyUser, (req,res,next) => {
    Order.findOne({user: req.user._id})
```

```
        .then((orders) => {
            if (!orders) {
                res.statusCode = 200;
                res.setHeader('Content-Type', 'application/json');
                return res.json({"orders": orders});
            }
            else {
                    res.statusCode = 200;
                    res.setHeader('Content-Type', 'application/json');
                    return res.json({"orders": orders});
            }

        }, (err) => next(err))
        .catch((err) => next(err))
    })
    .post(cors.corsWithOptions, authenticate.verifyUser, (req, res,
        next) => {
        Order.findOne({user: req.user._id}).then((order) => {
            if(!order) {
                Order.create({"user": req.user._id, "orders":
                    req.body}).then((order) => {
                    res.status = 200;
                    res.setHeader('Content-Type', 'application/json');
                    res.json(order);
                }, (err) => next(err));
            }
            else {
                    order.orders.push(req.body);
                    order.save().then((order) => {
                        res.status = 200;
                        res.setHeader('Content-Type', 'application/json');
                        res.json(order);
                    }, (err) => next(err));
            }
        }, (err) => next(err))
        .catch((err) => next(err));
    })
    .put(cors.corsWithOptions, authenticate.verifyUser, (req, res,
        next) => {
        res.statusCode = 403;
        res.end('PUT operation not supported on /orders/:cartId');
    })
    .delete(cors.corsWithOptions, authenticate.verifyUser, (req, res,
        next) => {
        Order.findOne({user: req.user._id}).then((order) => {
            if(order) {
                index = order.orders.indexOf(req.params.cartId);
                if(index >= 0) {
                    order.orders.splice(index, 1);
                    order.save().then((order) => {
                        res.status = 200;
```

```
                    res.setHeader('Content-Type', 'application/json');
                    res.json(order);
                }, (err) => next(err));
            }
            else {
                err = new Error(req.params.cartId + 'not found');
            err.status = 404;
            return next(err);
            }
        }
        else {
            err = new Error('Order not found');
            err.status = 404;
            return next(err);
        }
    }, (err) => next(err))
    .catch((err) => next(err));
});
module.exports = orderRouter;
```

3.2 Angular 8

3.2.1 App module

```
import { BrowserModule } from '@angular/platform-browser';
import { BrowserAnimationsModule } from
    '@angular/platform-browser/animations';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing/app-routing.module';
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { ProductsComponent } from './products/products.component';
import { ProductdetailComponent } from
    './productdetail/productdetail.component';
import { HomeComponent } from './home/home.component';
import { ProductService } from './services/product.service';
import { HttpClientModule } from '@angular/common/http';
import { baseURL } from './shared/baseurl';
import { CartComponent } from './cart/cart.component';
import { LoginComponent } from './login/login.component';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatListModule } from '@angular/material/list';
import { MatGridListModule } from '@angular/material/grid-list';
import { MatCardModule } from '@angular/material/card';
import { MatButtonModule } from '@angular/material/button';
import { MatDialogModule } from '@angular/material/dialog';
import { MatInputModule } from '@angular/material/input';
import { MatCheckboxModule } from '@angular/material/checkbox';
import { MatSelectModule } from '@angular/material/select';
import { MatSlideToggleModule } from '@angular/material/slide-toggle';
```

```
import { MatProgressSpinnerModule } from
    '@angular/material/progress-spinner';
import { MatSliderModule } from '@angular/material/slider';
import { MatFormFieldModule } from '@angular/material/form-field';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { AuthService } from './services/auth.service';
import { AuthInterceptor, UnauthorizedInterceptor } from
    './services/auth.interceptor';
import { AuthGuardService } from './services/auth-guard.service';
import { ProcessHTTPMsgService } from
    './services/process-httpmsg.service';
import { CartService } from './services/cart.service';
import {HTTP_INTERCEPTORS} from '@angular/common/http';
import { SignupComponent } from './signup/signup.component';
import { Ng2SearchPipeModule } from 'ng2-search-filter';
import { FooterComponent } from './footer/footer.component';
import { AddproductsComponent } from
    './addproducts/addproducts.component';
import { FileUploadModule } from 'ng2-file-upload';
import { ListproductsComponent } from
    './listproducts/listproducts.component';
import { AdminproductdetailComponent } from
    './adminproductdetail/adminproductdetail.component';
import { CheckoutComponent } from './checkout/checkout.component';
import { OrdersComponent } from './orders/orders.component';
import { ViewordersComponent } from './vieworders/vieworders.component';

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    ProductsComponent,
    ProductdetailComponent,
    HomeComponent,
    CartComponent,
    LoginComponent,
    SignupComponent,
    FooterComponent,
    AddproductsComponent,
    ListproductsComponent,
    AdminproductdetailComponent,
    CheckoutComponent,
    OrdersComponent,
    ViewordersComponent
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    AppRoutingModule,
    HttpClientModule,
    MatDialogModule,
```

```
    MatToolbarModule,
    MatListModule,
    MatGridListModule,
    MatCardModule,
    MatButtonModule,
    MatDialogModule,
    MatFormFieldModule,
    MatInputModule,
    MatCheckboxModule,
    MatSelectModule,
    MatSlideToggleModule,
    MatProgressSpinnerModule,
    MatSliderModule,
    FormsModule,
    ReactiveFormsModule,
    Ng2SearchPipeModule,
    FileUploadModule
  ],
  providers: [ProductService,
  {provide: 'baseURL', useValue: baseURL},
  ProcessHTTPMsgService,
  AuthService,
    AuthGuardService,
    CartService,
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptor,
      multi: true
    },
    {
      provide: HTTP_INTERCEPTORS,
      useClass: UnauthorizedInterceptor,
      multi: true
    }
    ],
  entryComponents: [
    LoginComponent,
    SignupComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3.2.2 Products Module

module.ts

```
import { Component, OnInit, Inject } from '@angular/core';
import { Product } from '../shared/product';
import { ProductService } from '../services/product.service';
import { Params, ActivatedRoute } from '@angular/router';
```

```
import { Location } from '@angular/common';
import { switchMap } from 'rxjs/operators';
import { CartService } from '../services/cart.service';
@Component({
  selector: 'app-products',
  templateUrl: './products.component.html',
  styleUrls: ['./products.component.css']
})
export class ProductsComponent implements OnInit {
   products : Product[];
   selectedProduct : Product;
  errMess: string;
  cart = false;
  quantity:number = 1;
  constructor(private productservice : ProductService,
      @Inject('baseURL') private baseURL,
  private cartservice : CartService) { }

  ngOnInit() {
   this.productservice.getProducts().subscribe(products =>
       this.products = products, errmess => this.errMess = <any>errmess);
  }
}
```

module.html

```
<!-- <div *ngFor = "let product of products" [routerLink] =
    "['/productdetail',product._id]">
   <h1>{{product._id}}</h1>
   <h2>{{product.name}}</h2>
</div> -->
<br><br><br>
<div style="text-align: center;">
<div class="title" id="product">WELCOME TO RAAGVASTRA</div>
<hr width="70%;">
  <div class="text" style="font-family: 'Montserrat', sans-serif;">
  Discover our curated range of exquisite handlooms and fabric... Do
      let us know if we can be of any help! Browse our featured
      collections below, or use the menu to explore all our
      products.<br><br>

</div>
<br>
<div class="title" id="banarasi">OUR PRODUCTS</div>
<hr width="40%;"></div>
<br>

<div class="container">

  <div class="row">
    <!--div class="col-md-1"></div-->
```

```
    <div class="col-md-4" *ngFor = "let product of products" >
      <div class="thumbnail">
        <img src="{{baseURL + 'images/' + product.image[0]}}"
            style="width:100%;height: 450px;">
        <div class="caption" ><br>
            <p> {{product.name}} <br></p>
            <p> &#8377;{{product.price}} </p>
            <button [routerLink] = "['/productdetail',product._id]"
                class="btn">View</button>
            <!-- <button (click) = "addToCart(product._id)" routerLink =
                "/cart" class="btn" style="float:right" >Add to
                cart</button> -->
        </div>
      </div>
    </div>
    </div>
</div>
```

3.2.3 Cart Module

module.ts

```
import { Component, OnInit, Inject } from '@angular/core';
import { Cart } from '../shared/cart';
import { CartService } from '../services/cart.service';
import { Product } from '../shared/product';
import { ProductService } from '../services/product.service';
import { CartProducts } from '../shared/cartproducts';
@Component({
  selector: 'app-cart',
  templateUrl: './cart.component.html',
  styleUrls: ['./cart.component.css']
})
export class CartComponent implements OnInit {
  cart: Cart;
  id: string;
  total: number = 0;
  products: any = [];
  product: Product;
  errMess: string;
  productids: string[];
  constructor(private cartservice : CartService,
    private productservice : ProductService,
   @Inject('baseURL') private baseURL) { }

  ngOnInit() {
   this.cartservice.getCart()
      .subscribe(cart => {
        this.cart = cart;
        this.id = cart._id;
        for(let j of cart.products)
```

```
        {
          this.total = this.total + j.price;
        }
        errmess => this.errMess = <any>errmess});
  }
  deleteCart(productid: string) {
    console.log(this.id);
    console.log(this.product);
    console.log('Deleting Product ' + this.id);
    this.cartservice.deleteCart(this.id,productid)
      .subscribe(cart => this.cart = <Cart>cart,
        errmess => this.errMess = <any>errmess);
  }

}
```

module.html

```
<br>
<br>
<br>
<div class="container-fluid" *ngIf = "cart || errMess">
   <!-- <div class="col-md-5" *ngFor = "let prod of products">
           <h1>{{prod.name}}</h1>
   </div> -->
   <div class="col-md-5 col-md-offset-2" *ngFor = "let product of
       cart.products">
      <h2>{{product.name}}</h2>
      <h2>{{product.quantity}}</h2>
      <h2>{{product.size}}</h2>
      <button (click)="deleteCart(product._id)" >Delete</button>
   </div>
   <h1>Total {{total}}</h1>
   <div *ngIf="errMess">
      <h2>Error</h2>
      <h4>{{errMess}}</h4>
   </div>
   <button routerLink = "/checkout">Buy</button>
</div>
```

3.2.4 Admin Module

module.ts

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { ProductService } from '../services/product.service';
import { Product } from '../shared/product';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-addproducts',
```

```
  templateUrl: './addproducts.component.html',
  styleUrls: ['./addproducts.component.css']
})
export class AddproductsComponent implements OnInit {
  filesToUpload: Array<File> = [];
  images: string[] = [];
  @ViewChild('pform') commentFormDirective;
  product = {name: '', description: '', price: '',image: [],xsmall: 0,
      small: 0, medium: 0,large: 0,xlarge: 0,xxlarge: 0,xxxlarge: 0};
formErrors = {
    'comment': ''
  };

  validationMessages = {
    'name': {
      'required':     'Name is required.'
    },
    'description': {
      'required': 'Description is required.'
    },
    'quantity': {
      'required': 'Quantity is required.'
    }
  };

  productForm: FormGroup;
  constructor(private productservice : ProductService,private fb:
      FormBuilder) { }

  ngOnInit() {
  }
  upload() {
    const formData: any = new FormData();
    const files: Array<File> = this.filesToUpload;

    for(let i =0; i < files.length; i++){
        formData.append("uploads[]", files[i], files[i]['name']);
        this.images.push(files[i].name);
    }
    // this.product.image = this.images;
    this.productservice.postImage(formData)
      .subscribe(files => console.log('files', files));

  }
  onSubmit()
  {
    this.product.image = this.images;
    console.log(this.product);
    this.productservice.postProduct(this.product)
      .subscribe(product => console.log('product', product));
  }
```

```
  fileChangeEvent(fileInput: any) {
    this.filesToUpload = <Array<File>>fileInput.target.files;
    //this.product.photo = fileInput.target.files[0]['name'];
  }
}
```

module.html

```
<br>
<br>
<br>
<br>
<div class="container">
    <input id="cin" name="cin" type="file"
        (change)="fileChangeEvent($event)" placeholder="Upload a
        file..." multiple/>
    <button type="submit" mat-button class="btn" (click) =
        "upload()">Upload Files</button>
    <form novalidate #productForm="ngForm" (ngSubmit)="onSubmit()">
        <mat-form-field>
            <input matInput placeholder="Name" type="text"
                [(ngModel)]="product.name" name="name" #name="ngModel"
                required>
            <mat-error *ngIf="name.errors?.required">Name is
                required</mat-error>
        </mat-form-field>
        <mat-form-field>
            <input matInput placeholder="Description" type="text"
                [(ngModel)]="product.description" name="description"
                #description="ngModel" required>
            <mat-error *ngIf="description.errors?.required">Description
                is required</mat-error>
        </mat-form-field>
        <mat-form-field>
            <input matInput placeholder="Price" type="text"
                [(ngModel)]="product.price" name="price" #price="ngModel"
                required>
            <mat-error *ngIf="price.errors?.required">Price is
                required</mat-error>
        </mat-form-field>
        <mat-form-field>
            <input matInput placeholder="XSmall" type="number"
                [(ngModel)]="product.xsmall" name="xsmall"
                #xsmall="ngModel" required>
            <mat-error *ngIf="xsmall.errors?.required">Quantity is
                required</mat-error>
        </mat-form-field>
        <mat-form-field>
            <input matInput placeholder="Small" type="number"
                [(ngModel)]="product.small" name="small" #small="ngModel"
```

```
                required>
            <mat-error *ngIf="small.errors?.required">Quantity is
                required</mat-error>
        </mat-form-field>
        <mat-form-field>
            <input matInput placeholder="Medium" type="number"
                [(ngModel)]="product.medium" name="medium"
                #medium="ngModel" required>
            <mat-error *ngIf="medium.errors?.required">Quantity is
                required</mat-error>
        </mat-form-field>
        <mat-form-field>
            <input matInput placeholder="Large" type="number"
                [(ngModel)]="product.large" name="large" #large="ngModel"
                required>
            <mat-error *ngIf="large.errors?.required">Quantity is
                required</mat-error>
        </mat-form-field>
        <mat-form-field>
            <input matInput placeholder="XLarge" type="number"
                [(ngModel)]="product.xlarge" name="xlarge"
                #xlarge="ngModel" required>
            <mat-error *ngIf="xlarge.errors?.required">Quantity is
                required</mat-error>
        </mat-form-field>
        <mat-form-field>
            <input matInput placeholder="XXLarge" type="number"
                [(ngModel)]="product.xxlarge" name="xxlarge"
                #xxlarge="ngModel" required>
            <mat-error *ngIf="xxlarge.errors?.required">Quantity is
                required</mat-error>
        </mat-form-field>
        <mat-form-field>
            <input matInput placeholder="XXXLarge" type="number"
                [(ngModel)]="product.xxxlarge" name="xxxlarge"
                #xxxlarge="ngModel" required>
            <mat-error *ngIf="xxxlarge.errors?.required">Quantity is
                required</mat-error>
        </mat-form-field>
        <button type="submit" mat-button class="btn"
            [disabled]="productForm.invalid">Submit</button>
    </form>
    <button class="btn" routerLink = "/viewproducts">View All
        Products</button>
</div>
```

3.2.5 Product Service

```
import { Injectable } from '@angular/core';
import { Product } from '../shared/product';
```

```
import { of,Observable } from 'rxjs';
import { HttpClient } from '@angular/common/http';
import { baseURL } from '../shared/baseurl';
import { map, catchError } from 'rxjs/operators';
import { ProcessHTTPMsgService } from './process-httpmsg.service';

@Injectable({
  providedIn: 'root'
})
export class ProductService {

  constructor(private http : HttpClient, private processHTTPMsgService
      : ProcessHTTPMsgService) { }
  getProducts(): Observable<Product[]> {
   return this.http.get<Product[]>(baseURL + 'products')
    .pipe(catchError(this.processHTTPMsgService.handleError));
  }
  getProduct(id: string): Observable<Product> {
    return this.http.get<Product>(baseURL + 'products/' + id)
    .pipe(catchError(this.processHTTPMsgService.handleError));
  }
  getFeaturedProduct(): Observable<Product> {
    return this.http.get<Product[]>(baseURL +
        'products?featured=true').pipe(map(products => products[0]))
    .pipe(catchError(this.processHTTPMsgService.handleError));
  }
  getProductIds(): Observable<string[] | any> {
   return this.getProducts().pipe(map(products => products.map(product
      => product._id)))
  .pipe(catchError(this.processHTTPMsgService.handleError));
  }
  postImage(images: File[]) {
    return this.http.post(baseURL + 'imageUpload/',images)
    .pipe(catchError(error =>
        this.processHTTPMsgService.handleError(error)));
  }
  postProduct(product: any): Observable<any> {
    return this.http.post(baseURL + 'products/',{'name': product.name,
        'description': product.description, 'price': product.price,
        'image': product.image, 'xsmall': product.xsmall, 'small':
        product.small, 'medium': product.medium, 'large': product.large,
        'xlarge': product.xlarge, 'xxlarge': product.xxlarge,
        'xxxlarge': product.xxxlarge})
    .pipe( map(res => {
      return {'success': true, 'name': product.name };
   }),
   catchError(error => this.processHTTPMsgService.handleError(error)));
 }
 putProduct(product: any): Observable<any> {
  return this.http.put(baseURL + 'products/' + product._id,{'name':
      product.name, 'description': product.description, 'price':
```

```
      product.price, 'xsmall': product.xsmall, 'small': product.small,
      'medium': product.medium, 'large': product.large, 'xlarge':
      product.xlarge, 'xxlarge': product.xxlarge, 'xxxlarge':
      product.xxxlarge})
  .pipe( map(res => {
    return {'success': true, 'name': product.name };
 }),
 catchError(error => this.processHTTPMsgService.handleError(error)));
}
  postComment(productId: string, comment: any) {
    return this.http.post(baseURL + 'products/' + productId +
        '/comments', comment)
    .pipe(catchError(this.processHTTPMsgService.handleError));
  }
  deleteProduct(id: string) {
    return this.http.delete(baseURL + 'products/' + id)
    .pipe(catchError(error =>
        this.processHTTPMsgService.handleError(error)));
  }
}
```

3.2.6 Cart Service

```
import { Injectable } from '@angular/core';
import { Cart } from '../shared/cart';
import { CartExists } from '../shared/cartExists';
import { HttpClient } from '@angular/common/http';
import { Observable, of } from 'rxjs';
import { baseURL } from '../shared/baseurl';
import { ProcessHTTPMsgService } from './process-httpmsg.service';
import { AuthService } from './auth.service';
import { map, catchError } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class CartService {

  constructor(private http: HttpClient,
    public auth: AuthService,
    private processHTTPMsgService: ProcessHTTPMsgService) { }

  getCart(): Observable<Cart> {
    if (!this.auth.isLoggedIn()) {
      return null;
    }
    return this.http.get<Cart>(baseURL + 'cart')
      .pipe(catchError(error =>
          this.processHTTPMsgService.handleError(error)));
  }
```

```
  postCarts(productids: any) {
    return this.http.post(baseURL + 'cart/', productids)
    .pipe(catchError(error =>
       this.processHTTPMsgService.handleError(error)));
  }

  isCart(id: string): Observable<CartExists> {
    if (!this.auth.isLoggedIn()) {
      return of({ exists: false, cart: null });
    }
    return this.http.get<CartExists>(baseURL + 'cart/' + id)
    .pipe(catchError(error =>
       this.processHTTPMsgService.handleError(error)));
  }

  postCart(name: string,description: string,quantity: number,size:
      string,price: number) {
    return this.http.post(baseURL + 'cart/', {'name':
       name,'description': description, 'quantity': quantity, 'size':
       size, 'price': price})
    .pipe(catchError(error =>
       this.processHTTPMsgService.handleError(error)));
  }

  deleteCart(id: string,productid: string) {
    return this.http.delete(baseURL + 'cart/' + id + '/products/' +
       productid)
    .pipe(catchError(error =>
       this.processHTTPMsgService.handleError(error)));
  }
  // getProductIds(): Observable<number[] | any> {
  //   return this.getCart().pipe(map(cart => cart.products.map(product
      => product.productid)))
  //     .pipe(catchError(error => error));
  // }
}
```

3.3 Screenshots

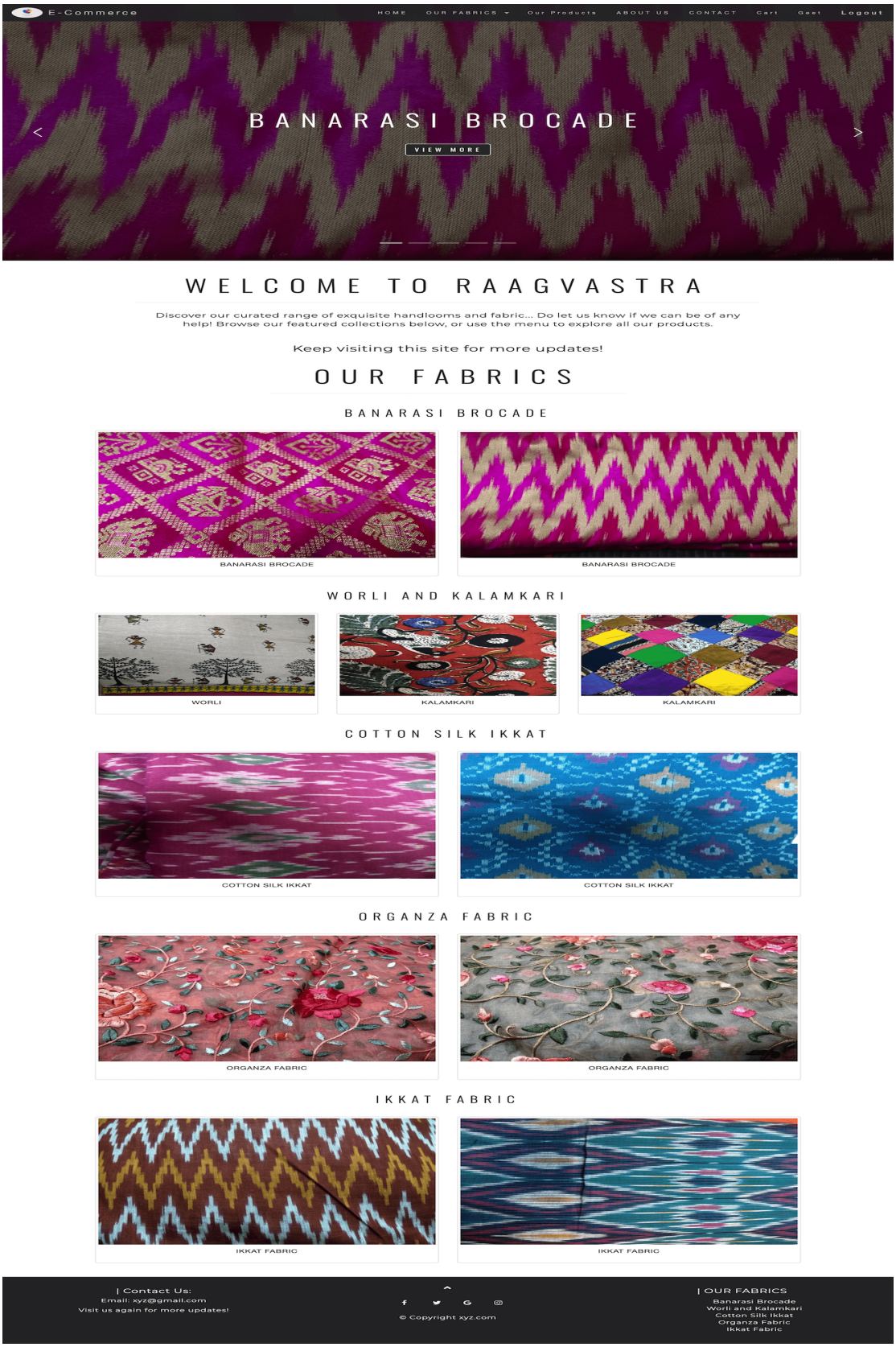

Figure 2: Home Page

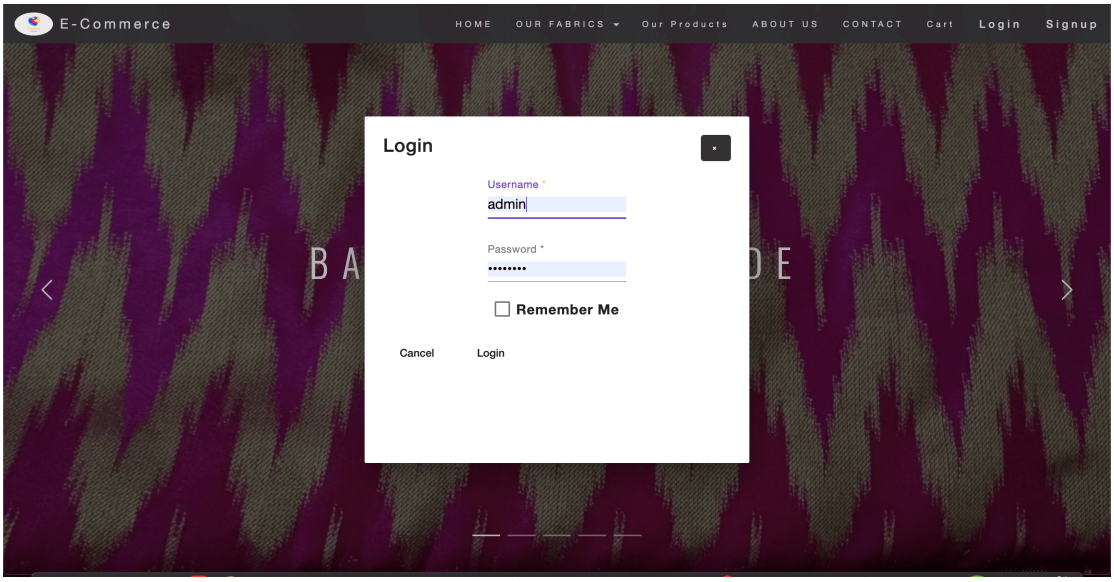

Figure 3: Login Modal

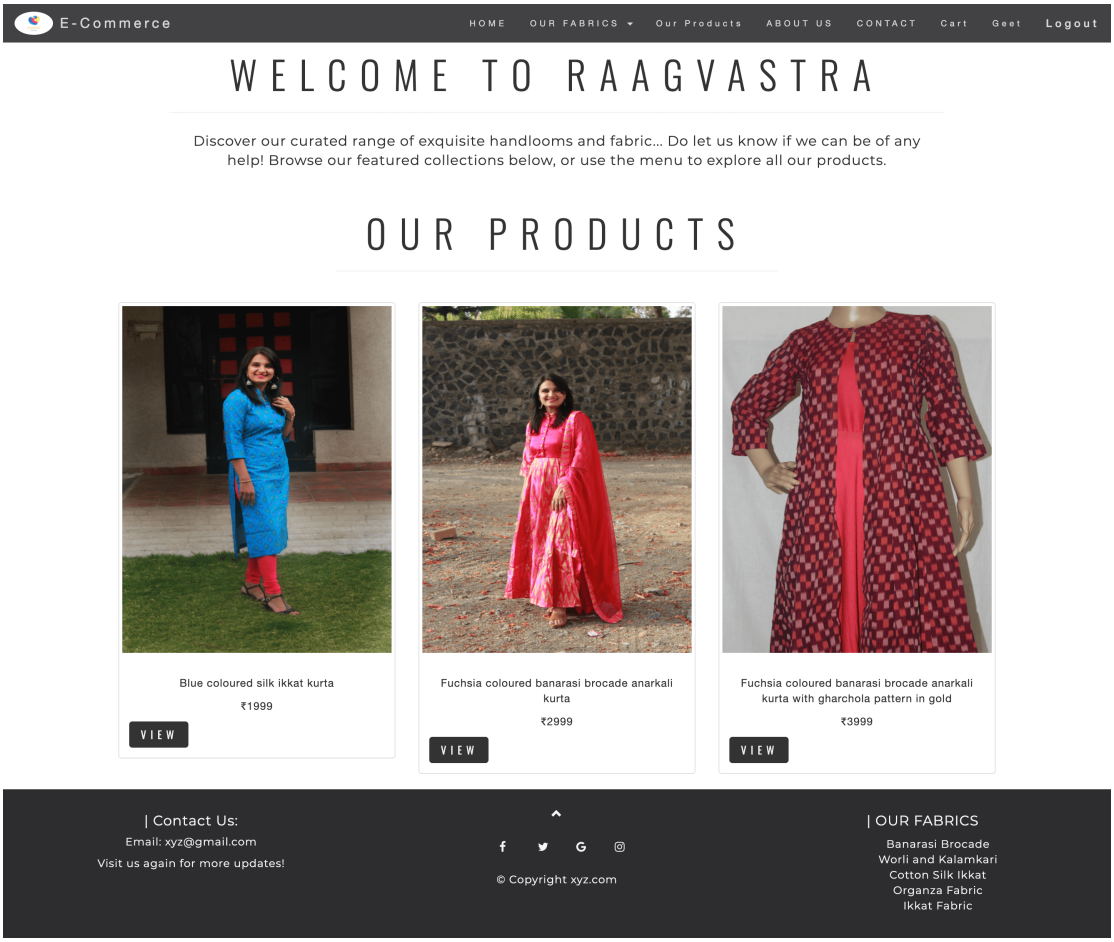

Figure 4: View Products Page

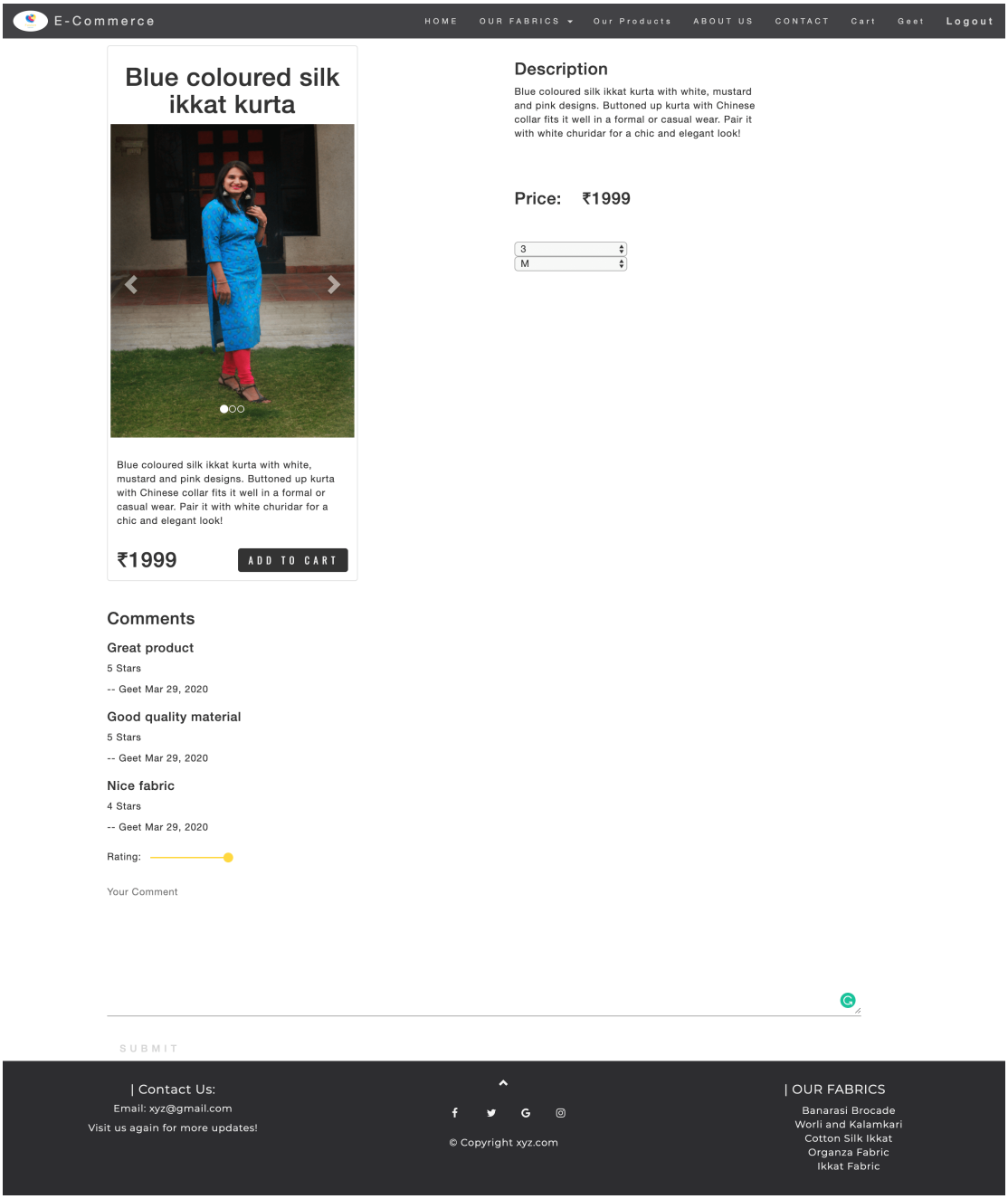

Figure 5: Product Detail Page

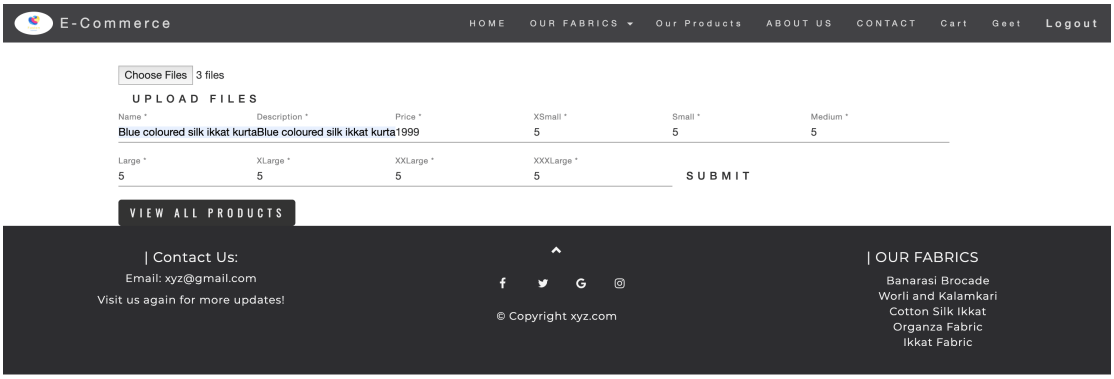

Figure 6: Admin Add Products Page

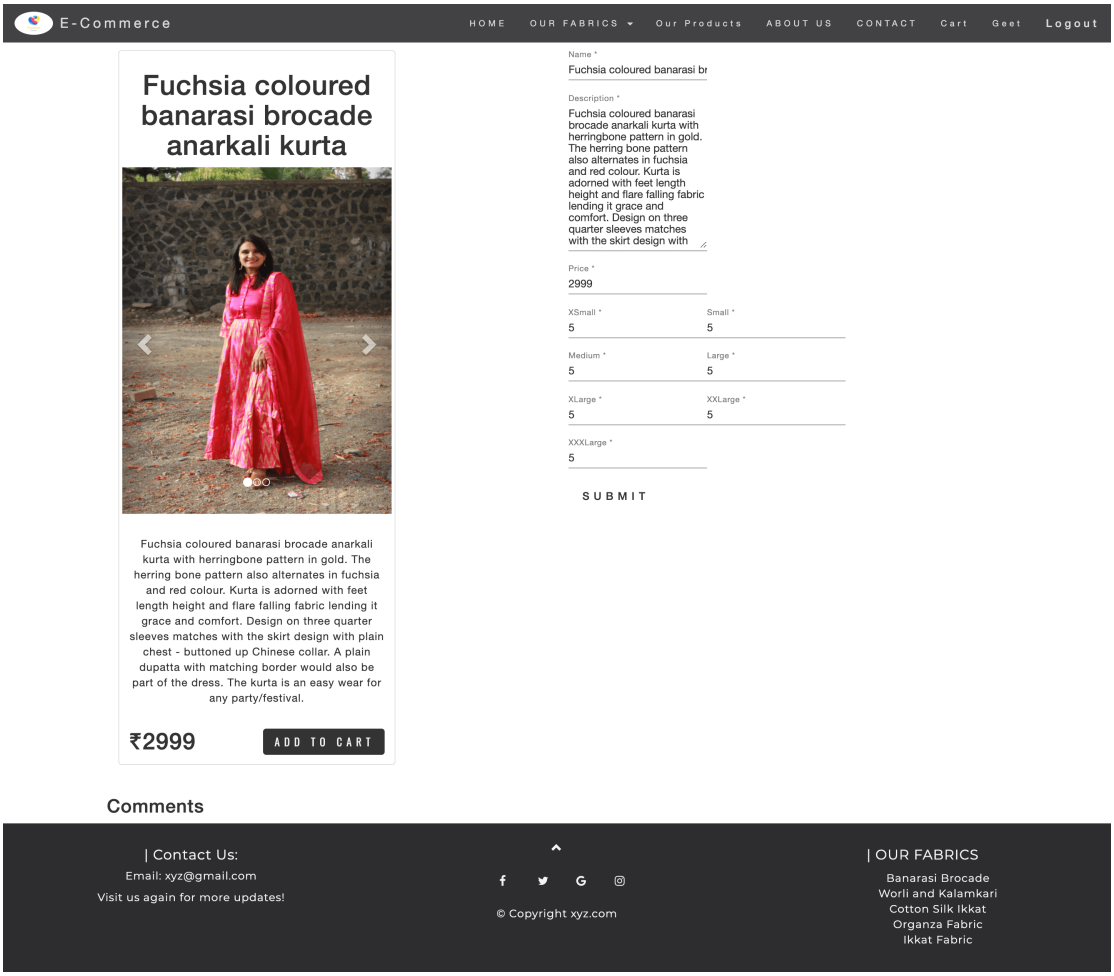

Figure 7: Admin Edit Product Page

4 DEPLOYMENT DETAILS

4.1 Building and serving from disk

During development, typically the *ng serve* command is used to build, watch, and serve the application from local memory, using webpack-dev-server. When ready to deploy, however, use of the *ng build* command is done to build the app and deploy the build artifacts elsewhere.
Both *ng build* and *ng serve* clear the output folder before they build the project, but only the *ng build* command writes the generated build artifacts to the output folder.
Nearing the end of the development process, serving the contents of the output folder from a local web server gives a better idea of how the application will behave when it is deployed to a remote server. Two terminals are required to get the live-reload experience.

- On the first terminal, the *ng build* command is run in watch mode to compile the application to the dist folder.

  ng build –watch

  Like the *ng serve* command, this regenerates output files when source files change.

- On the second terminal, a web server is installed (such as lite-server), and run it against the output folder. For example: *lite-server –baseDir="dist/project-name"*

The server automatically reloads the browser when new files are output.

4.2 Deployment to a remote server

A production build is created and the output directory to a web server is copied.

- For the production build:

  ng build –prod

- Copy everything within the output folder (dist/ by default) to a folder on the server.

- Configure the server to redirect requests for missing files to index.html

5 TESTING DETAILS

Figure 8: Testing of each Component

6 CONCLUSION

The project entitled E-COMMERCE APPLICATION FOR GARMENTS was completed successfully.The system has been developed with much care and free of errors and at the same time it is efficient and less time consuming. The purpose of this project was to develop a web application for purchasing items from a shop.This project helped us in gaining valuable information and practical knowledge on several topics like designing web pages using HTML,CSS,and management of database using MongoDB. The entire system is secured.Also the project helped us understanding about the development phases of a project and software development life cycle(SDLC). We learned how to test different features of a project.This project has given us great satisfaction in having designed an website which can be implemented to any nearby shops or branded shops selling various kinds of products by simple modifications.

FUTURE SCOPE:-

There is a scope for further development in our project to a great extent.A number of features can be added to this system in future like providing moderator more control over products so that each moderator can maintain their own products.System may keep track of history of purchases of each customer and provide suggestions based on their history. These features can be implemented in future to increase functionality of the project.

References

[1] http://www.w3schools.com/html/defualt.asp
http://www.w3schools.com/css/default.asp
http://www.w3schools.com/js/default.asp

[2] https://angular.io/

[3] https://expressjs.com/

[4] https://mongoosejs.com

[5] https://nodejs.org/en/

[6] https://www.myntra.com/

[7] JavaScript Enlightenment,Cody Lindley-First Edition, based onJavaScript 1.5, ECMA-262, Edition.