ONLINE SHOPPING



**A Project Report Submitted**

**In Partial Fulfillment of the Requirements for the Degree of**

MASTER OF COMPUTER APPLICATION

**by**

**PRAJJWAL SINGH**

**(1802914011)**

**Under the Supervision of**

**Dr. SANGEETA ARORA**

**KIET Group of Institutions**

**to the**

**FACULTY OF Computer Applications**

**DR. APJ ABDUL KALAM TECHNICAL UNIVERSITY LUCKNOW**

**(Formerly Uttar Pradesh Technical University, Lucknow)**

**July 2021**

# DECLARATION

I hereby declare that the work presented in this report entitled “Online Shopping", was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.

I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

Name: Prajjwal Singh

Roll No.: 1802914011

Branch: Master of Computer Application

**(Candidate Signature)**



VIN/HRD/2021/39 27th July 2021

**To whomsoever it may concern**

This is to certify that **Prajjwal Deep Singh,** student of **AKTU**, has successfully completed his Practice School II at **ValueCoders Services LLP** from **5th April 2021 to 27th July 2021** with the designation Jr. Associate Software Developer.

During this period, he was hardworking and dedicated to the tasks assigned. We wish him all the very best for a successful career ahead.

Yours Sincerely,

**For ValueCoders Services LLP**

Text, letter

Description automatically generated

**Lipi Mishra**

Sr - HR Manager (L & D)

# CERTIFICATE

Certified that **Prajjwal Singh (1802914011)** have carried out the project work having “**Online Shopping**” for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU**)** (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

**Dr. SANGEETA ARORA External Examiner**

Associate Professor

Department of Computer Applications

KIET Group of Institutions, Ghaziabad

**Dr. Ajay Shrivastava**

Professor & Head,

Department of Computer Applications KIET Group of Institutions, Ghaziabad

Date:

**Online Shopping**

**Prajjwal Singh**

**ABSTRACT**

The Online Shopping is a web-based application intended for online retailers. The main objective of this application is to make it interactive and its ease of use. It would make searching, viewing and selection of a product easier. It contains a sophisticated search engine for users to search for products specific to their needs. The search engine provides an easy and convenient way to search for products where a user can Search for a product interactively and the search engine would refine the products available based on the user’s input. The user can then view the complete specification of each product. They can also view the product reviews and write their own reviews. The application also provides a drag and drop feature so that a user can add a product to the shopping cart by dragging the item in to the shopping cart. The main emphasis lies in providing a user-friendly search engine for effectively showing the desired results and its drag and drop behavior.

# ACKNOWLEDGEMENTS

Success in life is never attained single handedly. My deepest gratitude goes to my thesis supervisor, **Dr. Sangeeta Arora** for her guidance, help and encouragement throughout my research work.

Words are not enough to express my gratitude to Dr. Ajay Kumar Shrivastava, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

**Prajjwal Singh**

# TABLE OF CONTENTS

Declaration i

Internship Certificate ii

Certificate iii

Abstract iv

Acknowledgements v

List of Figures x

[CHAPTER 1: INTRODUCTION 1](#_TOC_250017)

* 1. [Project Detail 2](#_TOC_250016)
  2. Purpose 2
  3. [Scope 2](#_TOC_250014)
  4. [Hardware / Software used in project 2-3](#_TOC_250013)
     1. [Hardware Used 2](#_TOC_250012)
     2. [Software Used 3](#_TOC_250011)

**CHAPTER 2: LITERATURE REVIEW 4-7**

2.1 Introduction 4

2.2 Literature Review 4

2.3 Major Conclusions 5-7

CHAPTER 3: TECHNICAL FEASIBILITY 8-15

1. [Technical Feasibility 8](#_TOC_250010)
   1. [Technology Description 8](#_TOC_250009)-15

3.2.1[Node.js 8-9](#_TOC_250008)

* + 1. [Angular (Typescript Framework) 9](#_TOC_250007)-13

3.2.2.1 [Features of Angular 8](#_TOC_250006)

* + - 1. [How Angular works? 9](#_TOC_250005)

3.2.2 [HTML 13](#_TOC_250004)

3.2.3 S[CSS 13](#_TOC_250003)

3.2.4 [JavaScript 14](#_TOC_250002)

3.2.5 [TypeScript 14](#_TOC_250001)

3.2.6 [Mongo DB 14](#_TOC_250000)

3.2.7 Express.js 15

**CHAPTER 4: BACKEND DESIGN 16-18**

* 1. Data Format 16-17
  2. Flowchart 18

**CHAPTER 5: FRONTEND DESIGN 19-27**

**CHAPTER 6: REPORT 28-52**

6.1 Coding 28-52

**CHAPTER 7: Testing 53-56**

* 1. Introduction 53
  2. Black Box Testing 54
  3. White Box Testing 54
  4. Unit Testing 55
  5. Integration Testing 56
  6. System Testing 56
  7. Acceptance Testing 56

**CHAPTER 8: LIMITATION 57**

* 1. Limitation 57
  2. Future Scope 57
  3. Future Enhancement 57

**CHAPTER 9: CONCLUSION 58**

**CHAPTER 10: REFERENCES 59-60**

**CHAPTER 11: BIBLIOGRAPHY 61**

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **Fig No** | **Title** | **Page No** |
| Fig 3.1 | MVC | 9 |
| Fig 3.2 | How Angular Works | 11 |
| Fig 4.1 | userinfo.model.ts | 14 |
| Fig 4.2 | Productinfo.model.ts | 15 |
| Fig 4.3 | Flowchart | 16 |
| Fig 5.1 | Home Component | 17 |
| Fig 5.2 | Searching in Home Component | 18 |
| Fig 5.3 | Sorting in Home Component | 19 |
| Fig 5.4 | Check if user is Logged in | 20 |
| Fig 5.5 | Login Form for User | 21 |
| Fig 5.6 | Registration Form for User | 22 |
| Fig 5.7 | OTP needs to be generated | 23 |
| Fig 5.8 | Home Component After user log in | 24 |
| Fig 5.9 | Checkout Component | 25 |
|  |  |  |

## CHAPTER 1 INTRODUCTION

Online shopping is the process whereby consumers directly buy goods or services from a seller in real-time, without an intermediary service, over the Internet. It is a form of electronic commerce. An online shop, shop, e-store, internet shop, web shop, online store, or virtual store evokes the physical analogy of buying products or services at a bricks-and-mortar retailer or in a shopping center. The process is called Business-to Consumer (B2C) online shopping.

An online shop evokes the physical analogy of buying [products](https://en.wikipedia.org/wiki/Product_(business)) or services at a regular ["bricks-and-mortar"](https://en.wikipedia.org/wiki/Brick_and_mortar_business) [retailer](https://en.wikipedia.org/wiki/Retailing) or [shopping center](https://en.wikipedia.org/wiki/Shopping_center); the process is called business-to-consumer (B2C) online shopping. When an online store is set up to enable businesses to buy from another businesses, the process is called [business-to-business](https://en.wikipedia.org/wiki/Business-to-business) (B2B) online shopping. A typical online store enables the customer to browse the firm's range of products and services, view photos or images of the products, along with information about the product specifications, features and prices.

Consumers find a product of interest by visiting the [website](https://en.wikipedia.org/wiki/Website) of the retailer directly or by searching among alternative vendors using a [shopping search engine](https://en.wikipedia.org/wiki/Shopping_search_engine), which displays the same product's availability and pricing at different e-retailers. As of 2020, customers can shop online using a range of different computers and devices, including [desktop computers](https://en.wikipedia.org/wiki/Desktop_computer), [laptops](https://en.wikipedia.org/wiki/Laptop), [tablet computers](https://en.wikipedia.org/wiki/Tablet_computer) and [smartphones](https://en.wikipedia.org/wiki/Smartphone).

Online stores usually enable shoppers to use "search" features to find specific models, brands, or items. Online customers must have access to the Internet and a valid [method of payment](https://en.wikipedia.org/wiki/Online_shopping#Payment) in order to complete a transaction, such as a [credit card](https://en.wikipedia.org/wiki/Credit_card), an [Interact](https://en.wikipedia.org/wiki/Interac)-enabled [debit card](https://en.wikipedia.org/wiki/Debit_card), or a service such as [PayPal](https://en.wikipedia.org/wiki/PayPal). For physical products (e.g., paperback books or clothes), the e-tailer ships the products to the customer; for digital products, such as [digital audio files](https://en.wikipedia.org/wiki/Digital_audio_file) of [songs](https://en.wikipedia.org/wiki/Song) or [software](https://en.wikipedia.org/wiki/Software), the e-tailer usually sends the file to the customer over the Internet. The largest of these online retailing corporations are [Alibaba](https://en.wikipedia.org/wiki/Alibaba_Group), [Amazon.com](https://en.wikipedia.org/wiki/Amazon.com), and [eBay](https://en.wikipedia.org/wiki/EBay).

### Project Detail

The shopping cart project needs to create the shopping cart system to organize the products record and the other information about the customers. How customers can buy products from website can be recognized from their username and password.

### Purpose

### The motive of this Online Shopping Web Application is to allow the user to play with the search tool and create different combinatorial search criterion to perform exhaustive search.

### • Making the application Node.js enabled so that multiple users can operate on a single thread, and response time of the server to each request will be reduced. Because of the reduced response time of the server more and more users will be able to shop at same time.

### • Provide Interactive interface through which a user can interact with different areas of application easily.

### • A search engine that provides an easy and convenient way to search for products specific to their needs. The search engine would list a set of products based on the search term and the user can further filter the list based on various parameters.

### • Provide the functionality for the user to add the products in Wishlist and cart only in a single click.

### Scope

Online shopping is rising day by day in India. Because India is the country where computer users are increasing day by day so as the online shopping trends are also increasing. This project covers the online selling of cosmetics, fashion accessories, watches etc. The project shows the product category and then product details. From the product details, the product can be added to cart and can be bought.

### Hardware / Software used in Project.

### Hardware Used

* + - * Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.70GHz
      * RAM: 4GB

### Software Used

* + - * Operating System: Windows 10
      * Text Editor: Visual Studio Code
      * Database: Mongo DB
      * Middleware: Express, NodeJS
      * Frontend: Angular

**CHAPTER 2**

**LITERATURE REVIEW**

1. **Introduction**

In the global business scenario, even the major retail players who are dominant in the market were finding themselves to find new offerings to serve their customers in a most delightful manner, so that they remain strong with higher satisfaction with greater loyalty. Ever since the emergence of online retailing, the situation of the on-store retailers in terms of sales promotional strategies to attract those customers into retail stores for purchase of products and services offered by them. Also, Shoppers buying orientation and buying Behavior had been drastically changed according to their expectations and situational factors. These shoppers have been given huge number of choices in terms of availability of a variety of products, brands and other major features associated with it, that has eventually made them to change themselves from the way of doing traditional shopping to the modern world of Online shopping. Another Important factor which paved the way for the shoppers towards online retail shopping was the technological transformation among the shoppers, where in these internet usages helped them in all the ways. These are some major influential factors which created a greater sophistication among the shoppers to go online retailing again and again. Here the researcher would like to do a detailed literature study on the shopper’s attitude and buying Behavior towards online shopping mode of purchasing products and services. This literature study would reveal the attitude of the online shoppers and at the same time the future researcher could be able to understand and find out the major factors affecting the buyer Behavior and quality decision making.

**2.2 Review of Literature**

**Simranjit Singh, Sonia Bajwa (2017)** had carried out a research study on “buying behavior of consumer towards counterfeit products: a case study of moga city”. The main objective of the research was to put emphasis on the reasons for counterfeit purchase and the norms they follow while forming this behavior. The other objectives were to find out the dominant factors affecting on the consumer while purchasing counterfeit goods, to know the relationship of these factors with the purchase behavior and application of TPB model for knowing the factors creating inclination towards counterfeit. The sample size was for research was 100 respondents. A structures questionnaire was used by following the convenience sampling method. The data were described with the help of pivot table and pie charts with percentages. After the analysis, the researcher had found that behavior of the respondents that leads them to go for committing a crime of purchasing such knock offs and pirated items. Cost and status seeking have known to be the two reasons for committing an illegal practice. If the money income of the consumer is increased, then this could work as a measure of curbing this practice.

**Priyanka Sharma (2017)** performed a research study on “Consumer Behavior towards Online Shopping-An Empirical Study with Reference To Bhiwani City, Haryana”. The main objective of the research was to identify the relationship of demographic factors that influence online shopping, to study the preferences of the consumers toward online shopping and the satisfaction level of the consumers while they shop online. To achieve the objective, null hypothesis „Online shopping is not reliable and trustworthy to the consumers‟ was tested. The sample size of the research was 100 respondents from Bhiwani City in Haryana state. The author had used Excel and Statistical Package for the Social Sciences (SPSS) software for data analysis purpose. After the data analysis and interpretation, the author concludes that the internet has given rise to great potential for businesses through connecting globally. The people having an annual income below Rs 2, 50,000 prefer cash on delivery and above Rs 2, 50,000 prefers Internet banking payments. Cash on delivery is found the most popular payment method for online shopping among consumers. Most of the people having annual income below Rs 4, 50,000 spends between 1500- 3000 per month for online shopping. People having annual income above Rs 4, 50,000 spends above 3000 per month for online shopping. The price of the products has the most influencing factor on online purchase and online shopping is getting popular in the younger generation. The Majority of the respondents buys from flipkart.com which is thus one of the leading online shopping websites in India. The main barrier is the safety of payment and privacy issue in the process of online shopping. This increases low levels of trust on online stores; therefore, sellers must make proper strategies to increase the consumer’s level of trust with them.

**Gagandeep Nagra, R. Gopal (2013)** performed a research study on “A study of Factors Affecting on Online Shopping Behavior of Consumers”. The main objective of the research was to identify the impact of Demographic factors impacting on-line shopping Behavior of consumers with special emphasis on Age, Gender, Education, Income Possession of internet, Frequency of online purchase, Motivation drives for online purchase. The sample size of the research was 70 respondents. The researcher had used “ANNOVA” method for the analysis of data. After the data analysis, the researcher concludes that the gender does impact Possession of internet and Frequency of online purchase of consumer’s occupation is a demographic variable which does not impact any of the variable. The respondents had perceived online shopping in a positive manner. On-line shopping in India is significantly affected by various Demographic factors like age, gender, marital status, family size and income.

**2.3 Major Observation and Conclusion**

The researcher would like to conclude through the detailed literature review study on Online Shopping become more famous among the customers because of the emergence of the internet usage among all the age categories. Even they started to do online shopping through mobile application which was made very much easier for them to do shopping in a better way. It was being identified that there was a drastic change in the shopping behavioral orientation among the customers. Even though the customers faced some challenges and difficulty in adapting the online mode of shopping, later they overcome these difficulties through the continuous feedback and customer services being provided to these online service providers. These Online service providers they do understand the minds of the customers and they provided these online platforms of shopping on an better reliability mode of shopping in order to save their time, instead they go for traditional ways of visiting the retail shop towards purchasing products available over there. Finally, the researcher would like to reveal that there is a broader scope of improvement in this area of study towards online shopping behavior, which could really be more interesting for the future researchers. Hence, Future researchers must focus on to the multiple aspects of online shopping in the global contest, which would be certainly an interesting fact and research to do so.

## CHAPTER 3

**TECHNICAL FEASIBILITY**

### Technical Feasibility

The system must be evaluated from the technical point of view first. The assessment of this feasibility must be based on an outline design of the system requirement in the terms of input, output, programs, and procedures. The project is developed within latest technology. The system has been developed using MEAN stack, so the project is technically feasible for development.

### Technology Description

## 3.2.1Node.js

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux [11].

Though .js is the standard [filename extension](https://en.wikipedia.org/wiki/Filename_extension) for JavaScript code, the name "Node.js" doesn't refer to a particular file in this context and is merely the name of the product. Node.js has an [event-driven architecture](https://en.wikipedia.org/wiki/Event-driven_architecture) capable of [asynchronous I/O](https://en.wikipedia.org/wiki/Asynchronous_I/O). These design choices aim to optimize [throughput](https://en.wikipedia.org/wiki/Throughput) and [scalability](https://en.wikipedia.org/wiki/Scalability) in web applications with many input/output operations, as well as for [real-time Web](https://en.wikipedia.org/wiki/Real-time_Web) applications (e.g., [real-time communication](https://en.wikipedia.org/wiki/Real-time_communication) programs and browser games).

Node.js operates on a [single-thread](https://en.wikipedia.org/wiki/Single_threading) [event loop](https://en.wikipedia.org/wiki/Event_loop), using [non-blocking I/O](https://en.wikipedia.org/wiki/Non-blocking_I/O) calls, allowing it to support tens of thousands of concurrent connections without incurring the cost of thread [context switching](https://en.wikipedia.org/wiki/Context_switch).[[70]](https://en.wikipedia.org/wiki/Node.js#cite_note-70) The design of sharing a single thread among all the requests that use the [observer pattern](https://en.wikipedia.org/wiki/Observer_pattern) is intended for building highly concurrent applications, where any function performing I/O must use a [callback](https://en.wikipedia.org/wiki/Callback_(computer_programming)). To accommodate the single-threaded event loop, Node.js uses the library—which, in turn, uses a fixed-sized thread pool that handles some of the non-blocking asynchronous I/O operations.

The features of Node.js are as follows:

**Asynchronous and Event Driven:** All APIs of Node.js library is asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.

**Very Fast:** Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.

**Single Threaded but Highly Scalable:** Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.[9]

**Buffering:** Node.js applications never buffer any data. These applications simply output the data in chunks.

### Angular (TypeScript Framework)

Angular is a TypeScript-based free and open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS.

### 3.2.2.1 Features of Angular

* **MVC Architecture**

[MVC](https://www.clariontech.com/blog/mvc-vs-flux-vs-redux-the-real-differences) stands for Model-View-Controller. The application data is managed by the Model and View manages the data display. While the controller plays as a connector between the view and model layers. Generally, in MVC architecture, you can split up the app accordingly and write code to connect the same.

### Efficient Two-Way Data Binding

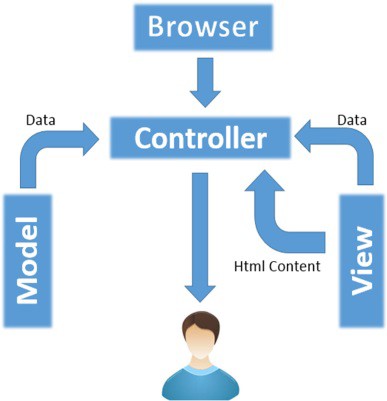
The most effective Angular feature is its two-way data binding system. The View layer exactly represents the Model layer, and they stay in perfect synchronization. If you make any change in the model, the users can see in the view model automatically. Thus, it decreases important development time.

### Dependency Injection

The built-in dependency injection of Angular makes application development easier for [developers](https://www.clariontech.com/blog/things-you-should-know-before-hiring-angular-developers). It just asks your dependencies. Just ask it ‘I need y’! And it creates the same & delivers it to you.

### How Angular works?

Angular is based on the Model-View-Controller (MVC) development pattern. The **Model-View-Controller (MVC)** is an architectural pattern that separates an application into three main logical components: the **model**, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects [13] [].



**Fig. 3.1 MVC**

* The **Model** represents your data structures. Typically, your model classes will contain functions that help you retrieve, insert and update information in your database.
* The **View** is information that is being presented to a user. A View will normally be a web page, but in CodeIgniter, a view can also be a page fragment like a header or footer. It can also be an RSS page, or any other type of page.
* The **Controller** serves as an intermediary between the Model, the View, and any other resources needed to process the HTTP request and generate a web page.

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your applications.

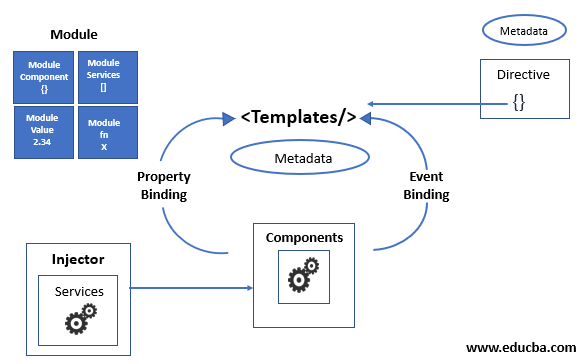
The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks of the Angular framework are Angular components that are organized into NgModules. NgModules collect related code into functional sets; an Angular application is defined by a set of NgModules. An application always has at least a root module that enables bootstrapping, and typically has many more feature modules.

* Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
* Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.

Modules, components and services are classes that use decorators. These decorators mark their type and provide metadata that tells Angular how to use them.

* The metadata for a component class associates it with a template that defines a view. A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display.
* The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).

An application's components typically define many views, arranged hierarchically. Angular provides the [Router](https://angular.io/api/router/Router) service to help you define navigation paths among views. The router provides sophisticated in-browser navigational capabilities.



**Fig. 3.2 How Angular Works**

## HTML

HTML is not a programming language, but it is a so-called markup language.

That is what the acronym HTML stands for: Hypertext Markup Language [1].

The core to all web design is Hypertext Markup Language (HTML), the code that sits behind every web page and allows users to create stunning websites. Today’s websites can do amazing things [2].

## SCSS

**Sass** (short for ***syntactically awesome style sheets***) is a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets (CSS). Sass Script is the scripting language itself [6]. The newer syntax, "SCSS" (Sassy CSS), uses block formatting like that of CSS. It uses braces to denote code blocks and semicolons to separate rules within a block. The indented syntax and SCSS files are traditionally given the [extensions](https://en.wikipedia.org/wiki/Filename_extension) .sass and .scss, respectively.

### JavaScript

JavaScript is a light-weight object-oriented programming language which is used by several websites for scripting the webpages [7]. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. JavaScript was initially created to “make web pages alive”. The programs in this language are called scripts. They can be written right in a web page’s HTML and run automatically as the page loads. Scripts are provided and executed as plain text. They do not need special preparation or compilation to run [15].

### TypeScript

TypeScript is a language created by Microsoft and released under an open-source Apache 2.0 License (2004). The language is focused on making the development of JavaScript programs scale to many thousands of lines of code. The language attacks the large-scale JavaScript programming problem by offering better design-time tooling, compile-time checking, and dynamic module loading at runtime [14].

## Mongo DB

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++.MongoDB is a cross-platform, document-oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document [10].

### Express.js

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications [12]. It facilitates the rapid development of Node based Web applications. Following are some of the core features of Express framework −

* Allows to set up middleware’s to respond to HTTP Requests.
* Defines a routing table which is used to perform different actions based on HTTP Method and URL.
* Allows to dynamically render HTML Pages based on passing arguments to templates.

## CHAPTER 4

**BACKEND DESIGN**

### User’s Format

Graphical user interface, text

Description automatically generated

**Fig 4.1 userinfo.model.ts**

* In this web application user will have the following details for the purpose of login credentials and user’s cart and Wishlist.

**4.2 Product Format**

A screenshot of a computer

Description automatically generated with medium confidence **4.2 productinfo.model.ts**

* In this web application a single product will have following information about itself. A user can search the product using these details.

### 4.3 Flowchart Diagram

Diagram

Description automatically generated

**Fig. 4.3 Flowchart of the applicatio**

## CHAPTER 5

**FRONTEND DESIGN**

**A screenshot of a cell phone

Description automatically generated with medium confidence**

**Fig 5.1 Home Component**

* This is the first page of interaction done by the user. When the user opens up the application this page will be open up first and with the help of this page user will be able to navigate through out the whole application.

**Graphical user interface, application

Description automatically generated**

**Fig 5.2 Searching in Home Component**

* Searching is done through pipes in angular, which is a vary crucial feature of angular framework, using pipes we can search for specific things at client side which reduces load from server.

**Graphical user interface

Description automatically generated**

**Fig 5.3 Sorting in Home Component**

* Sorting is done at the home page where user can sort the products according to its need. Various sorting features can be added over here.

**A screenshot of a cell phone

Description automatically generated with medium confidence**

**Fig 5.4 Check if User is logged in**

* If the user tries to access cart or add products into the cart a popup will appear to tell the user needs to log in first in the application.

**Graphical user interface, application

Description automatically generated**

**Fig 5.5 Login Form for User**

* User Login page, registered users will be able to login using this page. This form has client-side validation, which reduces the load on the server.

**A screenshot of a computer

Description automatically generated with medium confidence**

**Fig. 5.6 Registration Form for User**

* If a user is not registered, then that user needs to register itself first. This form also has client-side validation.

**Graphical user interface, application

Description automatically generated**

**Fig 5.7OTP needs to be generated**

* After registration, user will get an E-mail from the application with the help from Email JS, that email has the OTP for user’s verification.

**A screenshot of a cell phone

Description automatically generated with medium confidence**

**Fig 5.8 Home Component After User Login**

* This page is same as home component, but in the URL now there is a id which shows that a user is logged in right now.

**Graphical user interface, application, Teams

Description automatically generated**

**Fig 5.9 Checkout Component**

* After a user is done with its shopping, and want to check out, all the items in the cart will be shown with total price.

## CHAPTER 6

**REPORT**

1. **CODING**
   1. **Home Component**
      1. [**home.component.html**](http://home.component.html)

<mat-card class="mt-2 mb-1 mx-3">

    <div class="d-flex flex-row justify-content-around align-items-center">

        <div>

            <input class="search form-control" placeholder="Search Products Here" (keyup)="searchFunction($event)">

        </div>

        <div>

            <mat-form-field class="mt-0 pb-0">

                <mat-label>Sort By Price</mat-label>

                <mat-select (selectionChange)="sort($event)">

                    <mat-option value="none" selected>None</mat-option>

                    <mat-option value="high">Price: High to Low</mat-option>

                    <mat-option value="low">Price: Low to High</mat-option>

                </mat-select>

            </mat-form-field>

        </div>

        <div>

            <button mat-icon-button (click)="toCheckoutPage()">

                <span>To Cart</span><mat-icon matTooltip="To Cart" matBadge={{numberOfItemsInCart}} matBadgeColor="accent">shopping\_cart</mat-icon>

            </button>

        </div>

    </div>

</mat-card>

<div class="content">

    <div fxLayout="row wrap" fxLayoutGap="16px grid">

        <div fxFlex="25%" fxFlex.xs="100%" fxFlex.sm="33%" \*ngFor="let product of productInfo | filterPipe : searchText">

            <!--\*ngFor="let num of [1,2,3,4,5,6,7]" LOOP WILL BE HERE-->

            <mat-card class="mat-elevation-z4">

                <mat-card-header>

                    <mat-card-title>{{product.prodName}}</mat-card-title>

                    <mat-card-subtitle>{{product.prodBrand}}</mat-card-subtitle>

                </mat-card-header>

                <img mat-card-image src={{product.prodImage}}>

                <mat-card-content>

                    <p>{{product.prodDesc}}</p>

                    <p ><strong>Price: ₹</strong> {{product.prodPrice}}</p>

                </mat-card-content>

                <mat-card-actions class="d-flex justify-content-evenly align-items-center">

                    <button mat-mini-fab color="primary" matTooltip="Add to Wishlist">

                        <mat-icon>favorite</mat-icon>

                    </button>

                    <button mat-mini-fab color="primary" matTooltip="Buy">

                        <mat-icon>payments</mat-icon>

                    </button>

                    <button mat-mini-fab type="button" color="primary" matTooltip="Add to Cart" (click)="addToCart(product.id, product)">

                        <mat-icon>shopping\_cart</mat-icon>

                    </button>

                </mat-card-actions>

            </mat-card>

        </div>

    </div>

</div>

* + 1. [**home.component.ts**](http://home.component.ts)

export class ProductComponent implements OnInit {

  //URLs

  public userUrl: any = "http://localhost:3000/userInfo";

  public productUrl: any = "http://localhost:3000/productInfo";

  //Route Variables

  public routeSub: Subscription;

  //Information Variables

  public userInfo: any;

  public productInfo: any;

  public currentUser: any;

  public currentUserId: any;

  public numberOfItemsInCart: any = 0;

  public searchText: any;

  public sortSelect: any;

  public tempUser: any;

  public tempProducts: any = [];

  constructor(

    public http: HttpClient,

    public \_route: ActivatedRoute,

    public router: Router,

    public inCartItem: ToCartService,

    public userData: GetDataService

    ) {

      this.searchText = "";

      this.sortSelect = "";

    }

    ngOnInit(): void {

      this.routeSub = this.\_route.params.subscribe(params => {

        this.currentUserId = params['id'];

        //console.log(this.currentUserId);

      })

      this.getData();

      if(localStorage.getItem("userLogggedIn")){

        this.numberOfItemsInCart = this.userInfo.userCart.length;

      }else{

        this.numberOfItemsInCart = 0;

      }

  }

  getData(){

    //Fetching User's Data

    this.http.get("http://localhost:3000/userInfo").subscribe(data=>{

      this.userInfo = data

    })

    //Fetching Product's Data

    this.http.get("http://localhost:3000/productInfo").subscribe(data=>{

      this.productInfo = data;

    })

  }

  sort(someValue){

    switch (someValue.value){

      case "low":{

        this.productInfo = this.productInfo.sort((low, high) => low.prodPrice - high.prodPrice)

        break;

      }

      case "high":{

        this.productInfo = this.productInfo.sort((low, high) => high.prodPrice - low.prodPrice)

        break;

      }

      default:{

        this.productInfo = this.productInfo.sort((low, high) => low.prodPrice - high.prodPrice)

        break;

      }

    }

    return this.productInfo;

  }

  searchFunction(event){

    //console.log(event.target.value);

    this.searchText = event.target.value;

  }

  addToCart(product, wholeProduct){

    if(localStorage.getItem("userLoggedIn")){

      //Getting the ID of current logged in user

      this.currentUser = localStorage.getItem("userLoggedIn");

      //Storing the info of logged in user

      this.tempUser = this.userInfo[this.currentUser - 1];

      //Pushing the item in the user's cart array

      this.tempUser.userCart.push(product);

      //Displaying the number of items in the cart every time an item is added.

      this.numberOfItemsInCart = this.tempUser.userCart.length;

      //Added Item is pushed into the database

      this.http.put("http://localhost:3000/userInfo/"+`${this.currentUser}`, this.tempUser).subscribe();

      alert("Item Added to cart")

    }

    else{

      alert("Please Login First If you want to buy something")

    }

  }

  toCheckoutPage(){

    if(localStorage.getItem("userLoggedIn")){

      this.router.navigateByUrl('cart/'+`${this.currentUserId}`);

    }

    else{

      alert("Please Login First If you want to buy something");

      this.router.navigateByUrl('');

    }

  }

  sendProduct(wholeProduct){

    if(localStorage.getItem("userLoggedIn")){

      this.inCartItem.addToCart(wholeProduct);

      alert("Item added to the Cart");

    }

    else{

      alert("Please Login First If you want to buy something")

    }

  }

}

* 1. **Login Component**
     1. **login.component.html**

<div class="container col-8">

    <div class="user-wrapper">

        <h1>LOGIN</h1>

        <div class="my-3">

            <button type="button" class="btn btn-primary" (click)="toRegister()">Register</button>

        </div>

        <div class="container col-6">

            <form [formGroup]="loginForm" (ngSubmit)="loginSubmit()">

                <div class="my-3">

                    <label class="form-label m-0">Email</label>

                    <input class="form-control" type="email" placeholder="Email" formControlName="loginEmail">

                    <div class="text-danger"

                        \*ngIf="(returnLoginFormControl.loginEmail.touched || isSubmitted) && returnLoginFormControl.loginEmail.errors?.required">

                        Email is Required

                    </div>

                    <div class="text-danger"

                        \*ngIf="returnLoginFormControl.loginEmail.touched && returnLoginFormControl.loginEmail.errors?.email">

                        Enter Valid Email

                    </div>

                </div>

                <div class="my-3">

                    <label class="form-label m-0">Password</label>

                    <input class="form-control" type="password" placeholder="Password" formControlName="loginPwd">

                    <div class="text-danger"

                        \*ngIf="(returnLoginFormControl.loginPwd.touched || isSubmitted) && returnLoginFormControl.loginPwd.errors?.required">

                        Password is Required

                    </div>

                </div>

                <div class="my-3">

                    <button class="btn btn-primary" type="submit">User Login</button>

                </div>

            </form>

        </div>

    </div>

</div>

* + 1. **login.component.ts**

export class LoginComponent implements OnInit {

  public isSubmitted: Boolean = false;

  public userInfo: any;

  public userUrl = "http://localhost:3000/userInfo";

  public userId: any;

  constructor(

    public formBuilder: FormBuilder,

    public http: HttpClient,

    public getData: GetDataService,

    public router: Router

  ) {

    //Start - Login Form

    this.loginForm = this.formBuilder.group({

      loginEmail: ['', [Validators.required, Validators.email]],

      loginPwd: ['', [Validators.required]],

    });

    //End - Register Form

  }

  ngOnInit(): void {

    //window.location.reload();

    this.getData.getUserData().subscribe(response => {

      setTimeout(() => {

        this.userInfo = response

      }, 2000);

    })

    setTimeout(() => {

      console.log(this.userInfo);

    }, 3000);

  }

  //Start - Variables, Validation, FormControl & Submit Function of Login Form

  public loginForm: FormGroup;

  loginSubmit() {

    this.isSubmitted = true;

    let exist = true

    //console.log(this.loginForm.value)

    if (this.loginForm.valid) {

      exist = this.checkFunction(exist, this.loginForm.value.loginEmail);

      if (!exist) {

        localStorage.setItem("userLoggedIn", this.userId)

        this.router.navigateByUrl('home'+'/'+this.userId)

      } else {

        alert("Kindly Register Yourself First")

      }

    }

  }

  get returnLoginFormControl() {

    return this.loginForm.controls;

  }

  checkFunction(someBooleanValue, inputLoginValue) {

    for (let user of this.userInfo) {

      if ((user.userEmail === inputLoginValue) && user.isVerified) {

        this.userId = user.id;

        someBooleanValue = false;

        break;

      }

    }

    return someBooleanValue;

  }

  toRegister(){

    this.router.navigateByUrl('register');

  }

* 1. **Registration Component**
     1. **registration.component.html**

<div class="container col-8">

    <div class="admin-wrapper">

        <h1>REGISTER</h1>

        <div class="my-3">

            <div class="my-3">

                <button type="button" class="btn btn-primary" (click)="toLogin()">Login</button>

            </div>

            </div>

        <div class="container col-6">

            <form [formGroup]="registerForm" (ngSubmit)="registerSubmit()">

                <div class="form-group my-3">

                    <label class="form-abel m-0">Name</label>

                    <input id="userName" class="form-control" type="text" placeholder="Enter Name"

                        formControlName="userName">

                    <div class="text-danger"

                        \*ngIf="(returnFormControl.userName.touched || isSubmitted) && returnFormControl.userName.errors?.required">

                        Name is Required

                    </div>

                </div>

                <div class="form-group my-3">

                    <label class="form-abel m-0">Email</label>

                    <input id="userEmail" class="form-control" type="text" placeholder="Enter Email"

                        formControlName="userEmail">

                    <div class="text-danger"

                        \*ngIf="(returnFormControl.userEmail.touched || isSubmitted) && returnFormControl.userEmail.errors?.required">

                        Email is Required

                    </div>

                    <div class="text-danger"

                        \*ngIf="returnFormControl.userEmail.touched && returnFormControl.userEmail.errors?.email">

                        Enter Valid Email

                    </div>

                </div>

                <div class="form-group my-3">

                    <label class="form-abel m-0">Password</label>

                    <input class="form-control" type="password" placeholder="Enter Password" formControlName="userPwd">

                    <div class="text-danger"

                        \*ngIf="(returnFormControl.userPwd.touched || isSubmitted) && returnFormControl.userPwd.errors?.required">

                        Password is Required

                    </div>

                </div>

                <div class="form-group my-3">

                    <label class="form-abel m-0">Confirm Password</label>

                    <input class="form-control" type="password" placeholder="Confirm Password"

                        formControlName="userConfirmPwd">

                    <div class="text-danger"

                        \*ngIf="(returnFormControl.userConfirmPwd.touched || isSubmitted) && returnFormControl.userConfirmPwd.errors?.required">

                        Confirm Password is Required

                    </div>

                    <div class="text-danger"

                        \*ngIf="returnFormControl.userConfirmPwd.touched && returnFormControl.userConfirmPwd.errors?.mustMatch">

                        Password Does not Match

                    </div>

                </div>

                <div class="form-group my-3">

                    <label class="form-abel m-0">Phone Number</label>

                    <input class="form-control" type="number" placeholder="Phone Number" formControlName="userPhone">

                    <div class="text-danger"

                        \*ngIf="(returnFormControl.userPhone.touched || isSubmitted) && returnFormControl.userPhone.errors?.required">

                        Phone Number is Required

                    </div>

                    <div class="text-danger"

                        \*ngIf="returnFormControl.userPhone.touched && returnFormControl.userPhone.errors?.pattern">

                        Enter Valid Phone Number

                    </div>

                </div>

                <div class="form-group my-3">

                    <label class="form-abel m-0">Address</label>

                    <input id="userAdd" class="form-control" type="text" placeholder="Enter Address"

                        formControlName="userAdd">

                    <div class="text-danger"

                        \*ngIf="(returnFormControl.userAdd.touched || isSubmitted) && returnFormControl.userAdd.errors?.required">

                        Address is Required

                    </div>

                </div>

                <div class="form-group my-3">

                    <label class="form-abel m-0">PIN Code</label>

                    <input class="form-control" type="number" placeholder="Enter PIN Code of Your Area"

                        formControlName="userPin">

                    <div class="text-danger"

                        \*ngIf="(returnFormControl.userPin.touched || isSubmitted) && returnFormControl.userPin.errors?.required">

                        Pin Code is Required

                    </div>

                    <div class="text-danger"

                        \*ngIf="returnFormControl.userPin.touched && returnFormControl.userPin.errors?.pattern">

                        Enter Valid Pin Code

                    </div>

                </div>

                <div class="my-3">

                    <button class="btn btn-primary" type="submit">Register</button>

                </div>

            </form>

        </div>

    </div>

</div>

* + 1. **registration.component.ts**

export class RegistrationComponent implements OnInit {

  public isSubmitted: Boolean = false;

  public userInfo: any;

  public userUrl: any = "http://localhost:3000/userInfo";

  constructor(

    public formBuilder: FormBuilder,

    public http: HttpClient,

    public getData: GetDataService,

    public router: Router

  ) {

    //Start - Register Form

    this.registerForm = this.formBuilder.group({

      userName: ['', Validators.required],

      userEmail: ['', [Validators.required, Validators.email]],

      userPwd: ['', [Validators.required]],

      userConfirmPwd: ['', [Validators.required]],

      userPhone: ['', [Validators.required, Validators.pattern("^((\\+91-?)|0)?[0-9]{10}$")]],

      userAdd: ['', Validators.required],

      userPin: ['', [Validators.required, Validators.pattern('^[1-9]{1}[0-9]{2}[0-9]{3}$')]],

      userCartTry: [[]],

      userCart: [[]],

      userWishList: [[]],

      isLoggedIn: false,

      isVerified: false

    }, {

      validator: this.checkIfMatchingPasswords('userPwd', 'userConfirmPwd')

    });

    //End - Register Form

  }

  ngOnInit(): void {

    this.getData.getUserData().subscribe(response => {

      this.userInfo = response

    })

    setTimeout(() => {

      console.log(this.userInfo, 'register');

    }, 3000);

  }

  //Start - Variables, Validation, FormControl & Submit Function of Register Form

  public registerForm: FormGroup;

  checkIfMatchingPasswords(password: string, passwordConfirm: string) {

    return (formGroup: FormGroup) => {

      const mainPassword = formGroup.controls[password];

      const toMatchPassword = formGroup.controls[passwordConfirm];

      if (mainPassword.value !== toMatchPassword.value) {

        toMatchPassword.setErrors({ mustMatch: true })

      }

      else {

        toMatchPassword.setErrors(null)

      }

    }

  }

  get returnFormControl() {

    return this.registerForm.controls;

  }

  registerSubmit() {

    this.isSubmitted = true;

    let exist = false;

    if (this.registerForm.valid) {

      exist = this.checkFunction(exist, this.registerForm.value.userEmail)

      if (!exist) {

        this.http.post(this.userUrl, this.registerForm.value).subscribe();

        alert("Registered")

        this.router.navigateByUrl('verify');

      } else {

        alert("Email already exist");

      }

    }

  }

  checkFunction(someBooleanValue, inputEmail) {

    for (let user of this.userInfo) {

      if (user.userEmail === inputEmail) {

        someBooleanValue = true;

        break;

      }

    }

    return someBooleanValue;

  }

  toLogin(){

    this.router.navigateByUrl('login');

  }

  //End - Variables, Validation, FormControl & Submit Function of Register Form

* 1. **Verify Component**
     1. **verify.component.html**

<div class="container col-10">

    <div class=" w-50 mx-auto">

        <form id="f1" (submit)="sendEmail($event)" class="my-3">

            <h2>Send an OTP to Verify Your Email Mentioned Below</h2>

            <label class="form-label my-0">Email</label>

            <input class="form-control" type="email" name="email" [(ngModel)]="mail" required>

            <label [hidden]="true">OTP</label>

            <input [hidden]="true" name="message" [(ngModel)]="inputOtp">

            <label [hidden]="true">AppName</label>

            <input [hidden]="true" name="appName" [(ngModel)]="appName">

            <label [hidden]="true">Name</label>

            <input [hidden]="true" name="some\_user" [(ngModel)]="some\_user">

            <button type="submit" class="btn btn-primary my-3">Send OTP</button>

        </form>

        <form id="f2" class="my-2">

            <label class="form-label my-0">Enter OTP</label>

            <input class="form-control" type="password" placeholder="Enter OTP" name="otp" [(ngModel)]="otp" required>

            <button type="button" class="btn btn-primary my-3" (click)="verify()">Verify</button>

        </form>

    </div>

</div>

* + 1. **verify.component.ts**

export class VerifyComponent implements OnInit {

  public userUrl: any = "http://localhost:3000/userInfo";

  public userInfo: any;

  public thatValue: any;

  public mail: string;

  public some\_user: any;

  public inputOtp: any;

  public otp: any;

  public appName: any = "ShopApp";

  constructor(

    private http: HttpClient,

    private router: Router,

    private getUserData: GetDataService

  ) {

    this.some\_user = "";

    this.mail = "";

    this.inputOtp = Math.floor(Math.random() \* (9999 - 1000 + 1)) + 1000;

  }

  ngOnInit(): void {

    this.getUserData.getUserData().subscribe(response => {

      this.userInfo = response

    })

    setTimeout(() => {

      console.log(this.userInfo, 'verify');

    }, 3000);

  }

  checkFunction(someBooleanValue) {

    for (let user of this.userInfo) {

      if (user.userEmail === this.mail) {

        someBooleanValue = true;

        break;

      }

    }

    return someBooleanValue

  }

  public sendEmail(e: Event) {

    let exist = false;

    exist = this.checkFunction(exist)

    if (exist) {

      console.log(this.inputOtp);

      //EMAIL SENDING CODE COMMENTED TO SAVE EMAILS UNCOMMENT WHILE SHOWING

      e.preventDefault();

      emailjs.sendForm('service\_hmf1brw', 'template\_tiw81rh', e.target as HTMLFormElement, 'user\_e2APVICPX2gilPF7lkoEA')

        .then((result: EmailJSResponseStatus) => {

          console.log(result.text);

          alert("OTP Sent Please Check your Mail")

        }, (error) => {

          console.log(error.text);

        });

      //EMAIL SENDING CODE COMMENTED TO SAVE EMAILS UNCOMMENT WHILE SHOWING

    }

    else {

      if (exist == false) {

        alert("please enter Registered Email ID");

      }

      if (this.mail === "") {

        alert("please enter email id");

      } else {

        console.log("none");

      }

    }

  }

  verify() {

    if (this.otp == this.inputOtp) {

      alert("You are Verified Now, You Will be Redirected to Login Page");

      this.otp = "";

      this.userInfo.forEach(element => {

        if ((this.mail === element.userEmail) && (element.isVerified == false)) {

          element.isVerified = true;

          //console.log(element);

          this.http.put("http://localhost:3000/userInfo/" + element.id.toString(), element).subscribe((data) => { })

          this.router.navigateByUrl('login');

        }

      })

    }

  }

* 1. **Cart Component**
     1. **Cart.component.html**

<div class="container mt-3">

    <div class="parent">

        <div class="div1 ">

            <mat-card class="mb-3" \*ngFor="let item of productsInCart"><!--\*ngFor="let num of [1,2,3,4,5,6,7]" LOOP WILL BE HERE-->

                <div class="row">

                    <div class="col">

                        <mat-card-header>

                            <mat-card-title>{{item.prodName}}</mat-card-title>

                            <mat-card-subtitle>{{item.prodBrand}}</mat-card-subtitle>

                        </mat-card-header>

                        <img mat-card-sm-image src={{item.prodImage}}>

                    </div>

                    <div class="col">

                        <mat-card-content>

                            <p>{{item.prodDesc}}</p>

                            <p> <strong>Seller:</strong>{{item.seller}}</p>

                            <p> <strong>Price:</strong>{{item.prodPrice}}</p>

                            <!--<div class="d-flex">

                                <button mat-mini-fab color="primary">

                                    <mat-icon>remove</mat-icon>

                                </button>

                                <input \*ngIf="visible" class="mx-2 form-control form-control-sm" placeholder="Quantity" readonly value="{{item.prodQuantity}}">

                                <button mat-mini-fab color="primary">

                                    <mat-icon>add</mat-icon>

                                </button>

                            </div>-->

                        </mat-card-content>

                    </div>

                </div>

            </mat-card>

        </div>

        <div class="div2 float-end">

            <mat-card>

                <mat-card-title>Grand Total</mat-card-title>

                <mat-card-content>

                    <p>38,255 ₹</p>

                    <button mat-raised-button color="primary">Buy</button>

                </mat-card-content>

            </mat-card>

        </div>

    </div>

</div>

* + 1. **Cart.component.ts**

import { Component, OnInit} from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { ToCartService } from '../../Services/to-cart.service';

@Component({

  selector: 'app-cart',

  templateUrl: './cart.component.html',

  styleUrls: ['./cart.component.css']

})

export class CartComponent implements OnInit {

  public inCartInfo: any;

  public productIds: any;

  public productIndex: any = -1;

  public currentUserInfo: any;

  public currentUserId: any;

  public productsInCart: any = []

  public productInfo: any;

  public grandTotal: any = 0;

  constructor(public http: HttpClient, public cartService: ToCartService

    ) {}

    ngOnInit(): void {

      this.currentUserId = localStorage.getItem("userLoggedIn");

      this.http.get("http://localhost:3000/userInfo/" + `${this.currentUserId}`).subscribe((userInfo) => {

        setTimeout(() => {

          this.currentUserInfo = userInfo;

          this.inCartInfo = this.currentUserInfo.userCart;

          this.showProducts(this.inCartInfo)

        }, 2000);

      })

    }

    showProducts(productInfo){

      var sum = 0;

      console.log(this.inCartInfo);

      productInfo.forEach(element => {

        this.http.get("http://localhost:3000/productInfo/"+`${element}`).subscribe((data)=>{

          setTimeout(() => {

            this.productsInCart.push(data);

            sum = sum + this.productsInCart.prodPrice;

            this.grandTotal = sum;

          }, 2000);

        })

      })

    }

    public items =  this.cartService.getItems();

    getData(){

      this.http.get("http://localhost:3000/productInfo").subscribe((data)=>{

        this.productInfo = data;

      })

    }

  }

* 1. **home-page.module.ts**

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { Routes, RouterModule } from '@angular/router';

import { ProductComponent } from './product/product.component';

import { CartComponent } from './cart/cart.component';

import { FilterPipePipe } from '../Pipes/filter-pipe.pipe'

import { LoginGuard } from '../Guards/login.guard';

import { MatCardModule } from '@angular/material/card';

import { MatFormFieldModule } from '@angular/material/form-field';

import { MatSelectModule } from '@angular/material/select';

import { MatIconModule } from '@angular/material/icon';

import { MatBadgeModule } from '@angular/material/badge';

import { MatButtonModule } from  '@angular/material/button';

import { MatTooltipModule } from '@angular/material/tooltip';

import { FlexLayoutModule } from '@angular/flex-layout';

const homeRoutes: Routes = [

  {

    path: 'cart/:id',

    component: CartComponent,

    canActivate: [LoginGuard]

  },

  {

    path: 'home/:id',

    component: ProductComponent,

    canActivate: [LoginGuard]

  },

  {

    path: 'home',

    component: ProductComponent

  },

  {

    path: '',

    redirectTo: 'home',

    pathMatch: 'full'

  },

];

@NgModule({

  declarations: [

    ProductComponent,

    FilterPipePipe,

    CartComponent

  ],

  imports: [

    CommonModule,

    MatCardModule,

    MatFormFieldModule,

    MatSelectModule,

    MatIconModule,

    MatBadgeModule,

    MatButtonModule,

    FlexLayoutModule,

    MatTooltipModule,

    RouterModule.forRoot(homeRoutes)

  ]

})

export class HomePageModule { }

* 1. **login.module.ts**

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { Routes, RouterModule } from '@angular/router';

import { ReactiveFormsModule, FormsModule } from '@angular/forms';

import { VerifyComponent } from './verify/verify.component';

import { RegistrationComponent } from './registration/registration.component';

import { LoginComponent } from './login/login.component';

import { MatInputModule } from '@angular/material/input';

const loginRoutes: Routes = [

  {

    path: 'login',

    component: LoginComponent

  },

  {

    path: '',

    redirectTo: 'login',

    pathMatch: 'full'

  },

  {

    path: 'verify',

    component: VerifyComponent

  },

  {

    path: 'register',

    component: RegistrationComponent

  }

]

@NgModule({

  declarations: [

    VerifyComponent,

    RegistrationComponent,

    LoginComponent

  ],

  imports: [

    CommonModule,

    MatInputModule,

    ReactiveFormsModule,

    FormsModule,

    RouterModule.forRoot(loginRoutes)

  ]

})

export class LoginModule { }

* 1. **login.service.ts**

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Router } from '@angular/router';

import { Observable } from 'rxjs';

@Injectable({

  providedIn: 'root',

})

export class LoginService {

  public userUrl: any;

  public productUrl: any = "http://localhost:3000/productInfo"

  public userInfo: any;

  public ifRegistered: boolean = false;

  public locationId: number;

  constructor(

    public http: HttpClient,

    public router: Router

  ) {

    this.userUrl = "http://localhost:3000/userInfo";

    this.getData();

  }

  ngOnInit() { }

  getData() {

    this.http.get("http://localhost:3000/userInfo").subscribe(data => {

      this.userInfo = data;

    })

  }

  registerCheckFunction(loginFormData) {

    this.getData();

    //console.log(this.userInfo);

    //console.log(loginFormData.loginEmail);

    this.userInfo.forEach(element => {

      if (loginFormData.loginEmail === element.userEmail) {

        //console.log(element.userEmail);

        this.locationId = element.id - 1;

        this.ifRegistered = true;

        //window.location.reload();

      }

    });

    if (!this.ifRegistered) { return false; }

  }

  loginService(loginFormData) {

    this.registerCheckFunction(loginFormData)

    if (this.ifRegistered) {

      if (this.userInfo[this.locationId].isVerified) {

        if (loginFormData.loginEmail === this.userInfo[this.locationId].userEmail && loginFormData.loginPwd === this.userInfo[this.locationId].userPwd) {

          //this.userInfo[this.locationId].isLoggedIn = true;

          localStorage.setItem("userLoggedIn", this.userInfo[this.locationId].id);

          //this.http.put(this.userUrl+'/'+this.userInfo[this.locationId].id.toString(), this.userInfo).subscribe((data)=>{ this.userInfo = data })

          this.router.navigateByUrl('home' + '/' + this.userInfo[this.locationId].id.toString())

        }

        else {

          alert("Wrong Credentials");

        }

      }

      else {

        alert("You are not verified");

        this.router.navigateByUrl('verify');

      }

    }

    else {

      console.log("none");

    }

  }

  getUserData():Observable<any>{

    return this.http.get(this.userUrl);

  }

}

* 1. **register.service.ts**

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Router } from '@angular/router';

@Injectable({

  providedIn: 'root'

})

export class RegisterService {

  public userUrl: any = "http://localhost:3000/userInfo";

  public productUrl: any = "http://localhost:3000/productInfo"

  public currentUserInfo: any;

  public userInfo: any;

  public toVerification: any;

  constructor(

    public http: HttpClient,

    public router: Router

    ) {this.userUrl= "http://localhost:3000/userInfo";

    this.getData();}

   ngOnInit() {

     this.getData();

     console.log();

   }

  getData(){

    this.http.get("http://localhost:3000/userInfo").subscribe(data => {

      this.userInfo = data;

      //console.log(this.userInfo);

    })

  }

  registerService(registerFormData)

  {

    this.userInfo.forEach(element => {

      if(element.userEmail === registerFormData.userEmail){

        alert("This email is already registered, please try to login");

        return false;

      }

    });

    this.http.post(this.userUrl, registerFormData).subscribe();

    this.router.navigateByUrl('verify');

  }

}

* 1. **get-data.service.ts**

import { Injectable } from '@angular/core';

import { HttpClient, HttpParams, HttpHeaders } from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { map, catchError } from 'rxjs/operators';

@Injectable({

  providedIn: 'root'

})

export class GetDataService {

  public userUrl: any = "http://localhost:3000/userInfo";

  public productUrl: any = "http://localhost:3000/productInfo";

  constructor(private http: HttpClient) { }

  getUserData(): Observable<any>{

    return this.http.get(this.userUrl);

  }

  getProductData(): Observable<any>{

    return this.http.get(this.productUrl);

  }

  getCurrentUserData(id): Observable<any>{

    return this.http.get(this.userUrl+'/'+id.toString());

  }

}

## CHAPTER 7

**TESTING**

After completion of project, testing is needed to confirm the system can handle most of the situation. The testing is completed by follow a series of instruction to test the system ability. Before releasing the system, testing must be carried out carefully to increase the quality of system and provides better user experience for the users [3].

Software testing is employed to verify that the requirements have been correctly implemented, and that the software is fit for purpose, as well as identifying defects present in the software. There are various types of testing that may be conducted including unit testing, integration testing, system testing, performance testing, and user acceptance testing [5].

Testing is an integral part of any System Development Life Cycle insufficient and interested application tends to crash and result in loss of economic and manpower investment besides user’s dissatisfaction and downfall of reputation.

“Software Testing can be looked upon as one among much process, an organization performs, and that provides the last opportunity to correct any flaws in the developed system. Software Testing includes selecting test data that have more probability of giving errors.” The first step in System testing is to develop the plan that all aspect of system .Complements, Correctness, Reliability and Maintainability.

Software is to be tested for the best quality assurance, an assurance that system meets the specification and requirement for its intended use and performance.

System Testing is the most useful practical process of executing the program with the implicit intention of finding errors that makes the program fail.

1. **Black Box (Functional) Testing:**

Testing against specification of system or components. Study it by examining its inputs and related outputs. Key is to devise inputs that have a higher likelihood of causing outputs that reveal the presence of defects. Use experience and knowledge of domain to identify such test cases. Failing this a systematic approach may be necessary. Equivalence partitioning is where the input to a program falls into several classes, e.g., positive numbers vs. negative numbers. Programs normally behave the same way for each member of a class. Partitions exist for both input and output. Partitions may be discrete or overlap. Invalid data (i.e., outside the normal partitions) is one or more partitions that should be tested.

Internal System design is not considered in this type of testing. Tests are based on requirements and functionality.

This type of test case design method focuses on the functional requirements of the software, ignoring the control structure of the program. Black box testing attempts to find errors in the following categories:

* Incorrect or missing functions.
* Interface errors.
* Errors in data structures or external database access.
* Performance errors.
* Initialization and termination errors.

**7.2 White Box (Structural) Testing:**

Testing based on knowledge of structure of component (e.g., by looking at source code). Advantage is that structure of code can be used to find out how many tests case need to be performed. Knowledge of the algorithm (examination of the code) can be used to identify the equivalence partitions. Path testing is where the tester aims to exercise every independent execution path through the component. All conditional statements tested for both true and false cases. If a unit has no control statements, there will be up to 2n possible paths through it. This demonstrates that it is much easier to test small program units than large ones. Flow graphs are a pictorial representation of the paths of control through a program (ignoring assignments, procedure calls and I/O statements). Use flow graph to design test cases that execute each path. Static tools may be used to make this easier in programs that have a complex branching structure. Tool’s support. Dynamic program analyzers instrument a program with additional code. Typically, this will count how many times each statement is executed. At end print out report showing which statements have and have not been executed. Problems with flow graph derived testing:

# Data complexity could not consider.

# We cannot test all paths in combination.

# In only possible at unit and module testing stages because beyond that complexity is too high.

This testing is based on knowledge of the internal logic of an application’s code. Also known as a Glass Box Testing .Internal software and code working should be known for this type of testing. Tests are based on coverage of code statements, branches, paths, conditions.

### 7.3 Unit Testing

Unit Testing is mainly to test the system’s individual part of function. The unit test is requiring inputting to the system with all of the possible input such as correct input, empty input, wrong format and others possible input. This is to ensure the system can validate the input as correct input or invalid input. The unit testing will be carrying out included user login system.

Unit testing is a practice wherein we automatically execute our code to check that it does what we think it should. With unit testing, we can build a possibly large collection of tests that can quickly be executed, for instance every time we change the code, to make sure everything that worked still does [4].

### Integration Testing

In Integration Testing, the individual software modules are combined and tested as a whole unit. The integration testing generally follows unit testing where each module is tested as a separate unit. The main purpose of the integration testing is to test the functional and performance requirements on the major items of the project. All the modules of the project developed individually would be combined and tested as a whole system in the integration testing [8].

### System Testing

The system testing is mainly done on the whole integrated system to make sure that the project has been developed meets all the requirements. The test cases for the system testing will be the combination of unit and integration tests.

### Acceptance Testing

This testing is generally performed when the project is nearing its end. This test mainly qualifies the project and decides if it will be accepted by the users of the system. The users or the customers of the project are responsible for the test.

## CHAPTER 8

**LIMITATION**

### Limitation

This application does not have payment gateway. An external checkout package must be integrated into this application. This application does not have features by which user can set price ranges for products and receive alerts once the price reaches the range.

### Future Scope

The following things can be done in future. Workshop management

* The users could subscribe for price alerts which would enable them to receive messages when price for products fall below a particular level.
* The current system is confined only to the shopping cart process. It can be extended to have an easy to use check out process.
* Users can have multiple shipping and billing information saved. During checkout they can use the drag and drop feature to select shipping and billing information.

### Future Enhancement

There are many things that can be done in this project such as adding a functionality which will tell the user which of the item is available to buy or not. When a product is already in the Wishlist, and its price drops the user will be notified via mail, and many more things, so there can be quite as many future enhancements are possible.

## CHAPTER 9

**CONCLUSION**

The ‘Online Shopping’ is designed to provide a web-based application that would make searching, viewing and selection of a product easier. The search engine provides an easy and convenient way to search for products where a user can Search for a product interactively and the search engine would refine the products available based on the user’s input. The user can then view the complete specification of each product. They can also view the product reviews and also write their own reviews. Use of Ajax components would make the application interactive and prevents annoying post backs. Its drag and drop feature would make it easy to use.

## CHAPTER 10

**REFERENCES**

1. Akshi Kumar, Web Technology, CRC Press, 1st Edition, PP 135.
2. Amos Q. Haviv, MEAN Web Development, Second Edition, PP 84
3. Gerald D. Everett, and Raymond McLeod, Jr, Software Testing, Testing Across the Entire Software Development Life Cycle, PP 93.
4. Gerald D. Everett, and Raymond McLeod, Jr, Software Testing, Testing Across the Entire Software Development Life Cycle, PP 51.
5. Gerard O’Regan, Concise Guide to Software Testing, Springer Nature Switzerland AG, 1st Edition, PP 49.
6. Iztok Fajfar, Start Programming Using HTML, CSS, and Javascript, CRC Press, 1st Edition, PP 2.
7. Iztok Fatjar, Start Programming Using HTML, CSS, and Javascript, CRC Press, 1st Edition, PP 38.
8. Martin P. Robillard, Introduction to Software Design with Java, Springer Nature Switzerland AG, 1st Edition, PP 91.
9. Matthew David, HTML5 Designing Rich Internet Applications, Focal Press, 2nd Edition, PP 3.
10. Mithun Sateesh, Bruno J. D’mello, and Jason Krol, Web Development with MongoDB and Node JS, 2nd Edition, PP 3.
11. Mithun Sateesh, Bruno J. D’mello, and Jason Krol, Web Development with MongoDB and Node JS, 2nd Edition, PP 9.
12. Olga Filipova, and Rui Vilao, Software Development from A to Z, PP 183.
13. Shyam Seshdri, Angular Up & Running, PP 14.
14. Steve Fenton, Pro TypeScript Application -Scale JavaScript Develoment, 1st Edition, PP xix.
15. William Turner, Javascript for Sound Artists, CRC Press, 1st Edition, PP 1.

## CHAPTER 11

**BIBLIOGRAPHY**

* + [www.w3schools.com](http://www.w3schools.com/)
  + [www.angular.io](https://angular.io/)
  + <https://material.angular.io/components/categories>
  + <https://developer.mozilla.org/en-US/>
  + [www.stackoverflow.com](http://www.stackoverflow.com/)