INDIRA GANDHI NATIONAL OPEN UNIVERSITY

MCSP - 060

Online Mobile Store

by

Rejoy Nair
Enrolment No: 137132696

Under Guidance
of
Mrs. Deepa Krishnan

Submitted to the School of Computer and Information Sciences, IGNOU
in partial fulfilment of the requirements
for the award of the degree
Master of Computer Applications (MCA)
2017

Indira Gandhi National Open University
Maidan Garhi
New Delhi – 110068.

SCHOOL OF COMPUTER AND INFORMATION SCIENCES
IGNOU, MAIDAN GARHI, NEW DELHI – 110 068

II. PROFORMA FOR THE APPROVAL OF MCA PROJECT PROPOSAL (MCSP-060)

(Note: All entries of the proforma of approval should be filled up with appropriate and complete information. Incomplete proforma of approval in any respect will be summarily rejected.)

Project Proposal No :.....
(for office use only)

Enrolment No.: 137132696
Study Centre: 0111
Regional Centre: Hyd RC Code: 01
E-mail: rejoy.nair@yahoo.com
Mobile/Tel No.: 9819507501

1. Name and Address of the Student: Rejoy Nair, S1, Gr Floor, plot 79/80, Madhavanagar colony, Miyapur, Hyderabad
2. Title of the Project***: 'online Mobile Store'
3. Name and Address of the Guide: Mrs Deepa krishnan, S1, plot 79/80, Madhavanagar Colony, Miyapur, Hyderabad
4. Educational Qualification of the Guide (Attach bio-data also)

Ph.D*	M.Tech.*	B.E*/B.Tech.*	MCA	M.Sc.*
☐	☑	☐	☐	☐

(*in Computer Science / IT only)

5. Working / Teaching experience of the Guide**: Over 8 years experience in teaching graduate and Post graduate students

(**Note: At any given point of time, a guide should not provide guidance for more than 5 MCA students of IGNOU)

6. Software used in the Project***: GoLang, Postgresql, HTML-CSS

(*** Please refer to section VIII of these guidelines)

7. If already pursued BCA/BIT from IGNOU, mention the title of the project (CS-76) and the s/w used: NA
8. Project title of the Mini Project (MCS-044) and the s/w used: 'Simulation of Data request to the cloud' (S/w- Java)
9. Is this your first submission? ☑ Yes ☐ No

Signature of the Student
Date: 8th Nov 2016

Signature of the Guide
Date: 08/11/2016

For Office Use Only

Approved	Not Approved
☑	☐

Name: K. Amit Kumar
27/2/17
Signature, Designation, Stamp of the Project Proposal Evaluator
Date: .....

Suggestions for reformulating the Project:

Bio-Data of Project Guide

DEEPA KRISHNAN
M. Tech (Information Security)

S1, Plot # 79/80, SreeVenkataLakshmi Nilayam,

Madhava Nagar Miyapur, Hyderabad, Telangana-500049

Mob: +91 9773423043 **email:** deepakrishnan84@gmail.com

SUMMARY:

Post graduate in Information Security having 8 years experience in teaching graduate and post graduate level students and with high level of expertise in Network Security and Computer Networks. Authored research papers which have featured in reputed journals and has initiated and guided research projects with colleagues and students. Key areas of interest include Wireless Networks, Cloud Computing, Data Analytics and Big data and Distributed Computing.

EXPERIENCE: **8 years**

Assistant Professor, Dept of Information Technology	**Feb 2016 – Present**
B.V. Raju Institute of Technology, JNTU, Hyderabad	

- Academic theory sessions for M.Tech and B.Tech students

Assistant Professor, Dept of Information Technology	**Dec 2011 – Aug 2011**
Pillai Institute of Information Technology, Media Studies and Research Centre , Mumbai University	

- Set up the Information and Network Security lab
- Lab-In Charge of IT Lab
- Project guide for undergraduate Computer Engineering and Information Technology course
- Special Topic Seminar guide for Undergraduate courses
- Handled theory and Practicals of Information and Network Security and System Security Course
- Mentor for College level Technical Paper Presentation Contest
- Interview Panel Member for Selecting Candidates for Masters Program

Lecturer (Computer Science, Information Security)	**Jan 2008 – May 2009**
Amrita Vishwa Vidya Peetham, Kerala, India	

- Instructor for various Graduate and Under Graduate for computer science courses like System Software and Operating Systems, Data Structures and Algorithms, Computer Organization and Architecture
- Designed course program and determine the evaluation policies and coordinating the question paper setting.
- University mentor and student guide for E-Commerce Course.

Lecturer (Computer Science)	**Jan 2006 – Jan 2008**
Sree Buddha College of Engineering, Kerala University	

- Lecturer for Computer Organization and Architecture, Multimedia Systems.
- Lab Instructor for C and C++ programming, Data Structures and Algorithms and Computer Organization and Architecture

OTHER WORK EXPERIENCE: **1 year**

Teaching Assistant	**Jul 2010 – Dec 2010**
SRM University, Chennai	

- Handled laboratory and tutorial sessions of Network Security for 1st year M. Tech students
- Volunteered and coordinated various university and department level conferences and workshops

Trainee Researcher at Resource Centre for Cyber Forensics (Symbian, C#, Android)	**Jan 2011 – Jun 2011**

CDAC (Gov of India's Center for Development of Advanced Computing)	Internship

- Proposed agent based forensic data extraction and analysis process and model for Symbian phones as part of the comprehensive Smartphone forensic tool
- Developed a smart phone forensic application for Symbian phones.
- Documented various Smart Phone OS capabilities and vulnerabilities
- Presented research finding at AICERA 2011 International Conference

EDUCATION:

Qualification / Year	University/ Board/ Institute	CGPA/ Percentage
M. Tech (Masters in Information Security & Computer Forensics) 2009 – 2011	SRM University, Chennai, India	9.57 / 10
B. Tech 2001- 2005	Cochin University, Kerala, India	71
AISSE CBSE [11th - 12th] 1999 – 2001	SN Central School, Kerala, India	83
AISSE CBSE [10th] 1999	SN Central School, Kerala, India	88.8

IT SKILL SET:

Languages:	C, C++, C#.NET,ASP.NET, Java, JavaScript, HTML5
Database:	Microsoft SQL Server, Oracle
Operating System:	Windows 7, Windows 2003, Linux
Networking Tools:	Cisco Packet Tracer, Network Simulator 2.34,Qualnet Simulator, PeerSimP2P Simulator, IP Traffic Test and Measure(ZTI),Net Disturb Impairment Emulator Software for IP Networks(ZTI), ZTI LAN Traffic Version 2.7 Traffic Generator for IP Networks, CloudSim
Computer Forensics:	Network Session Analyzer, Email Tracer, Cyber Check Suite, EnCase Enterprise Edition
Design Tools:	Visual Studio 2012
Training/Courses:	Certified Ethical Hacking Version 8 by EC Council Valid till August 2016 CompTIA Security Plus Valid from July 2013 to 2016 Cyber Criminology at University of Madras Workshop on Effective Research Paper Writing by IIT Bombay Certified Cloud Computing Engineer by Innobuzz Knowledge Solutions Certified Information Security Expert by Innobuzz Knowledge Solutions Workshop on IBM Cognos BI tool at TCS Powai

RESEARCH WORK:

Paper Publications:	Deepa Krishnan, *"A Distributed Intrusion Detection system for Mobile Ad hoc Networks using tamper evident Mobile Agents"*, ICICT 2014, Published in Elsevier Procedia Computer Science, www. sciencedirect.com Deepa Krishnan, Madhumita Chatterjee, *"Cloud Security Management Suite- Security As a Service"* , October 30 – Nov 2nd 2012 , WICT 2012, Trivandrum, India, Proceedings in IEEE Explore Deepa Krishnan, Madhumita Chatterjee, *"An Adaptive Intrusion Detection System for Cloud Computing Framework"*, SNDS 2012, Trivandrum, India, October 11-12, 2012. Proceedings in Springer Recent Trends in Computer Networks and Distributed Systems Security ISBN: 978-3-642-34134-2 (Print) 978-3-642-34135-9 (Online)

	Deepa Krishnan, Satheesh Kumar S, A.Arokiaraj Jovith, *"Symbian Phone Forensics-An Agent based Approach"*, AICERA 2011, Published in International Journal of Scientific and Engineering Research Volume3 April 2012
Membership of Professional Bodies:	The Indian Society for Technical Education Life Member (Membership No: LM 83949) The Computer Society of India (Membership No:01174929) The Cryptology Research Society of India (Membership No: L/5737)

Date: 05/03/2017

Signature

CERTIFICATE OF ORIGINALITY

This is to certify that the project report entitled ONLINE MOBILE STORE submitted to **Indira Gandhi National Open University** in partial fulfilment of the requirement for the award of the degree of **MASTER OF COMPUTER APPLICATIONS (MCA) ,** is an authentic and original work carried out by Mr. / ~~Ms.~~ REJOY NAIR with enrolment no. 137132696 under my guidance.

The matter embodied in this project is genuine work done by the student and has not been submitted whether to this University or to any other University / Institute for the fulfilment of the requirements of any course of study.

.....

Signature of the Student:

Date: 5th Mar 2017

Name and Address
of the student

Rejoy Nair ,
S/, Plot No 79/80 ,
Sree Venkata lakshmi Nilayam
Madhavanagar colony, Miyapur,
Hyderabad - 500049
Enrolment No.....137132696

.....

Signature of the Guide

Date: 05/03/2017

Name, Designation
and Address of the
Guide:

Deepa krishnan
Assistant Professor
Dept of IT,
BVRIT Narsapur
Medak District
Ph: +91 9773423043

Contents

1 INTRODUCTION / OBJECTIVES

Electronic Commerce (also e-commerce) commonly written as E-commerce or ecommerce is the trading or facilitation of trading using Computer Network such as Internet or Social Networks. E-commerce draws on technologies like Mobile Commerce, Electronics Funds Transfer, Supply Chain Management, Internet Marketing, Online Transaction Processing, Electronic Data Interchange (EDI), Inventory Management Systems and automated data collection systems. Modern electronic commerce typically uses the World Wide Web for at least one part of the transaction‘s life cycle though it may also use other technologies like email.

Online Shopping is a form of electronic commerce which allows consumers to directly buy goods and services from a seller over the Internet using a Web Browser. Consumers find a product of interest by visiting the website of the retailer directly or by searching among alternative vendors using a shopping Search Engine which displays the same product‘s availability and pricing at different vendors. Customers can shop online using a range of different computers and devices including desktop computers, laptops, tablet computers and smartphones.

1.1 Background

An online shop evokes the physical analogy of buying products and services at regular “bricks-and-mortar“ retailer or shopping center; the process is called Business-to-consumer (B2C) 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 product along with the information about the product specification, features and prices.

Online stores typically enable shoppers to use “search“ features to find specific models, Brands or Items. Online customers must have valid access to the Internet and a valid method of payment in order to complete a transaction, such as a credit card, an Interac-enabled debit card, or a service such as 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 of songs or software, the e-tailer typically sends the file to the customer over the Internet. The largest of these online retailing corporations are Alibaba, Amazon.com and eBay.

English entrepreneur Michael Aldrich was a pioneer of online shopping in 1979[1]. His system connected a modified domestic TV to a real-time transaction processing computer via a domestic telephone line. The first World Wide Web server and browser, created by Tim Berners-Lee in 1990, opened for commercial use in 1991. Thereafter, subsequent technological innovations emerged in 1994: online banking, the opening of an online pizza shop by Pizza Hut, Netscape SSL v2 encryption standard for secure data transfer and Intershop‘s first Online shopping system. The first retail transaction over the Web was ei-

ther by NetMarket or Internet Shopping Network in 1994. Immediately after, Amazon.com launched its online shopping site in 1995 and eBay was also introduced in 1995. Alibaba's sites Taobao and Tmall were launched in 2003 and 2003 respectively.

Mobile Phone online buying platforms can be broadly classified into 2 types

- Owned by Retailer to sell own products
- Marketplace, which allows various merchants to showcase and sell their products. The retailer only manages the marketplace.

This project describes a minimal implementation of a platform which the retailer can use to sell own products i.e. the retailer is accountable and responsible for the product inventory.

1.2 Purpose & Motivation

The main purpose of this project is to create an online store to buy mobile phones. The site will allow users to search mobile phones from the products listing page. Users can add selected products to a shopping cart and checkout by making payment. Users will receive an order copy of their invoice.

The retailer website will be managed by an Admin. Admin will have additional functionality such as managing product catalog and generating reports.

Motivation to work on this project includes

- Working on a project in the Retail domain
- To gain knowledge of the working of a good user friendly website that facilitates online transactions using a database
- Interest in technologies such as Golang, Javascript, HTML, CSS and SQL for web development
- Explore data analytics that can be implemented using Golang

1.3 Objectives

The Key objectives of the Project include

- Implementation of an Admin module for managing a website facilitating buying of mobile phones using online transactions.
- Develop and host a website which allows users to search and explore mobile phones
- Implement the shopping cart feature for the site that allows users to add selected products and tag it to a single order
- Implement the online payment module (Credit Cards Only)

- Explore technologies such as Golang, Javascript, HTML, CSS and SQL for web development
- Explore data analytics that can be implemented using Golang

1.4 Project Category

This project can be categorized as a web development project that uses concepts of Internet technologies and web design, web security and RDBMS. Though Golang is not an OOP language, OOP has been achieved by making use of interfaces allowed in Golang. Network security has also been implemented for the website using Transport Layer Security (TLS)

2 SYSTEM ANALYSIS

2.1 Identification of Need

Small scale retailers or start-up retailers in the mobile phone category would like to get their website up and running with a minimum investment in hardware and technology. The technology should be chosen in a way that allows to scale up later if required.

Large retailers may want to look at alternative technologies that make possible to lower development time and cost translating to better Project Management and increased efficiency.

Mobile Phones is an indicative category. The Online selling platform can be deployed for the purpose of selling almost any kind of product categories and services online. Mobile Phone Category has been chosen for this project due to the limited and universally well-understood feature set of the products of this category that allows to describe the project implementation without delving too much into the business aspects of the products.

2.2 Preliminary Investigation

India was one of the fastest growing retail e-commerce markets in 2015, growing at the rate of 129.5 per cent Y-o-Y. Declining broadband subscription prices and the launch of 4G services has become the driving forces of e-commerce in the country. India will see more people come online than any other country in the next 15 years. With the penetration of digital devices and social media in the interiors of the country, online sellers have been presented with an unprecedented opportunity of growth, becoming extremely attractive to investors. E-commerce is expected to acquire 4.8% market share in total retail sales by 2019.

Among e-tail categories, mobile phone and mobile accessories continue to be the top contributor to the overall pie[2]. Retailers would be willing to invest in technology and even maintain it in-house provided it is low-cost, easy to maintain and of course effective in serving their business needs. This allows retailers to manage their own product catalog, respond to market dynamics by carrying out promotional campaigns, manage the look and feel of their website and quickly go-live and roll-out the changes. Though all of this could also be achieved through an intermediate service provider, the retailer in that case would not be able to manage costs and time as they would like to.

2.3 Feasibility Study

The objective of a feasibility study is not to solve the problem but to acquire a sense of its scope. This project does not aim to build a full scale website that is ready for industrial deployment. The project aims to explore web development

using Golang by creation of an online mobile store. The full scale implementation of the project is constrained by time, resource and cost and is in fact not really necessary for an academic project of this kind.

The scope of the project shall be limited to

- User Registration and Sign-on
- Manage Products Catalog
- Order Management
    - Shopping Cart
    - Place Order
    - Cancel Order
- Payment Gateway Integration
- Notifications
- Analytics & Reports

To deploy a website with the basic benchmarks as stated above the following tools, platforms, hardware and software were used.

The development environment was set up on a i686 computer loaded with a 32-bit Linux operating system. The host environment shall be the same i686 computer with the 32-bit Linux operating system i.e., the development server and the host server are one and the same machine.

Later, after the development process is complete, the option of deploying the web app on a remote host or Google App Engine may be explored for demo purpose.

Table 2.1: **Software & Hardware Requirements**

Sr. No.	**Tools & Technologies**	**Description**
1.0	Go ver 1.7 linux/386	Tool for managing Go source code.Go (also commonly referred to as golang) is an open source systems programming language developed at Google

2.0	net/http package	Package http of Golang that provides http client and server implementations
3.0	SQLite3 Database	SQLite is a self-contained high-reliability, full-featured, public-domain, SQL database engine.
4.0	SQLite Manager	SQLite Manager is a Database Management System for SQLite database and is available as a firefox addon that can be used in the browser.
5.0	Emacs ver. 24.3	Development Environment
6.0	go-mode	Emacs package for GoLang
7.0	Geany ver 1.22	A lightweight IDE with support for HTML, CSS, Javascript and JQuery
8.0	Github	Repository Management Cloud
9.0	Git	Version Control System
10.0	HTML5	HTML 5 is the markup language used for structuring and designing content on the world wide web

11.0	CSS	A declarative stylesheet language for structured documents
12.0	Javascript	Javascript is a high-level, dynamic, untyped and interpreted programming language for front-end web functionality
13.0	JQuery ver 1.11.3	JQuery is a cross-platform javascript library designed to simplify the client side scripting of HTML
14.0	Crunchbang Linux Waldorf 11.0	Operating System
15.0	stripe-go	Golang package for Stripe API. Stripe is a Payment Gateway Service provider

2.4 Project Planning

Software Development

- The online Mobile Store shall be a minimalist functional website with functionalities as described in the scope
- Server Code shall be developed entirely using Golang
- Web server provided by Golang's "net/http" package shall be used to serve the pages
- Front-end development shall be done using HTML 5, CSS3, Javascript and JQuery.
- Shopping cart shall be developed entirely using javascript. The shopping cart will not be persistent and will be valid only for the User Session.
- Stripe Checkout shall be used for Payment Gateway. API integration shall be done with the Go code.

- Personal Software Process (PSP) Software Development Methodology shall be followed

Deployment & Hosting

- The website shall be hosted on the local server on which the development is done. Internet connectivity will be required for the web application to work
- Options to deploy the web app on Google Cloud App Engine shall be explored for demo purpose

Testing

- Test Scenarios, Test Cases & Testing

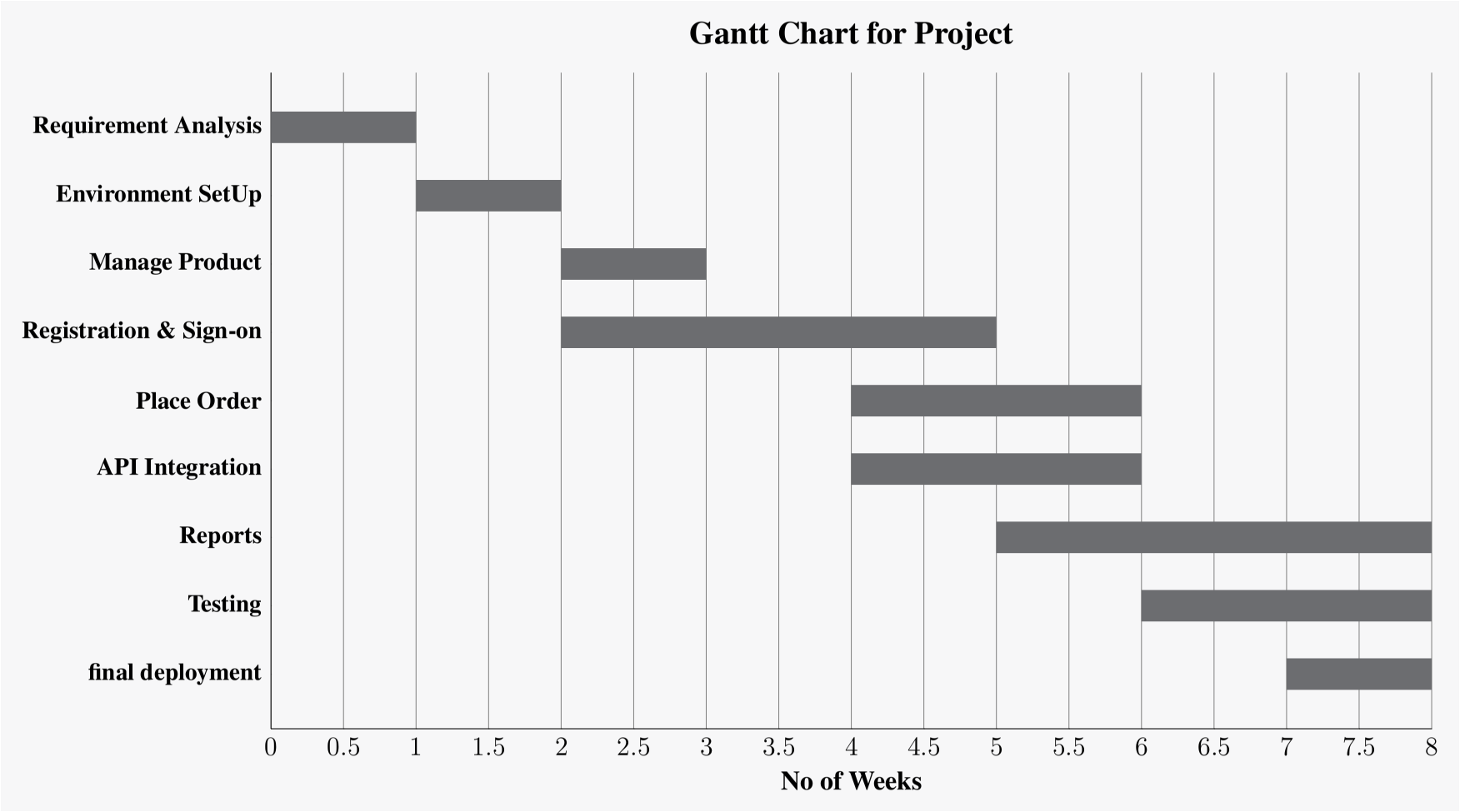
Resources & Cost

- This is an academic project and a single resource i.e., I myself shall be working on all aspects of the Project.
- There is no financial cost attributed to the project other than the renumeration bill for the Project Guide to be footed by IGNOU.

2.5 Project Scheduling

Major deliverables of this project are:

- Setting up the development environment
- Code implementation of key functions i.e Product Catalouge, Sign-On, Order management & Reports
- Payment API Integration
- Test Scenarios, Test Cases & Testing
- Deployment & Hosting


Gantt Chart for Project
Requirement Analysis
Environment SetUp
Manage Product
Registration & Sign-on
Place Order
API Integration
Reports
Testing
final deployment
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6
6.5
7
7.5
8
No of Weeks

2.6 Software Requirement Specifications

2.6.1 Problem Definition

Mobile Phone Retailer requires to present inventory to customers online and facilitate users to search, select and place orders for mobile phones as well as make payments online. Retailer should be able to manage the platform that allows the buying of mobile phones online.

2.6.2 User Module

Users can register, login, search for products, add selected products to shopping cart and place orders after check-out

2.6.3 User Module - Registration

Users can register on the customer website by clicking on a link that directs them to the Sign-up or registration form. Users have to provide the username with which they register, first and last name, an email address and confirm the password they want to create for the username

Table 2.2: **Requirements - User Module - Registration**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR1	User should be able to navigate to the sign-up / registration form from the landing page	Hyperlink	User should have a secure connection to the website
FR2	User should be able to create a unique username for registration purpose	Input text box String	should be unique should be alphanumeric

FR3	User should be able to create a provide First Name, Last name and email address for registration purpose. Email address should be unique	Input text boxes String	Should be string (alpha) Email Address should be a valid email ID and unique
FR4	User should be able to create a create a password for the username	Password box string	should be alphanumerc User is required to reconfirm
FR5	User should receive error alerts for invalid input	Alert boxes String	Empty space is not allowed Fields that require unique value shall only accept such values Fields not meeting the criteria described in the regex pattern for the field shall be invalidated

2.6.4 User Module - Sign-On

User can sign-on using the username and password

Table 2.3: **Requirements - User Module - Registration**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR6	User connection should be secure		Transport Layer Security (TLS) should be implemented Secure connection via HTTPS
FR7	User should be able to log into the website using the username and password created at the time of registration	Input text boxes String	User should authenticate by providing valid password New session shall be generated for every successful login
FR8	User should receive error alerts for invalid input	Alert boxes string	Username cannot be empty and should be valid Password should correspond to the correct password of the username maintained at the server.

2.6.5 User Module - Search Products

User can search for specific products in the product catalog by description. By default all products shall be listed in the main view.

Table 2.4: **Requirements - User Module - Product Listing & Search**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR9	All products should be listed in the main view. Listings should include the image, name and price of the product	Info Panel Image File, String	data object should be present in the product catalog
FR10	User should be able to search the product by entering string that shall be matched with the product description. User should be able to view only such products that fulfill the search criteria	Search Box string	Search string cannot be empty
FR11	User should be able to select a product by adding it to the shopping cart	Button Each product info panel shall have its own button	A single unit of the product shall be added to the shopping cart each time the button is clicked

2.6.6 Shopping Cart

Selected products can be added to shopping cart. Shopping Cart is not Persistent i.e. it is valid only for the session.

Table 2.5: **Requirements - User Module - Shopping Cart**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR12	User should be able to view the list of products added to the shopping cart. The list should include the image, name and price of the product.	Dropdown list (Tabular) image File, String	Details of the Product should be similar to the selected product in all aspects
FR13	User should be able to change the quantity of the product added to the shopping cart	Input Number Box Integer	Minimum value shall be 1
FR14	User should be able to view the total cost per product in the shopping cart	Output text Number	Total Cost cannot be Zero Total Cost = Quantity * Price Per Unit

FR15	User should be remove the product from the shopping cart	Button	Quantity should be decremented by 1 each time this button is clicked Entire line item should get removed if the product quantity is 1 when this button gets clicked. i.e., it is equivalent to removing the product from the shopping cart
FR16	User should be able to checkout and directly place an order from the shopping cart	Button	User session should be valid

2.6.7 Place Order

Users can trigger orders after checkout by making payment.

Table 2.6: **Requirements - User Module - Place Order**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR17	User should be able to view the list of products in the order basket. The list should include the image, name, quantity and price of the product.	Order Data info panel Image file, String, Integer	List of products should correspond to the ones added to the shopping cart Product details must be equal in all respects to the ones added to the shopping cart
FR18	User should be able to view the total cost per product in the order basket	Output Text Number	Total Cost cannot be Zero Total Cost = Quantity * Price Per Unit

FR19	User should be able to view the overall order quantity of products and the overall order cost	Output Text Integer, Number	Overall Order Quantity cannot be Zero. Overall order cost cannot be zero. Overall Order Quantity = Sum of total quantity of individual product items of the order basket Overall Total Cost = Sum of total cost of individual product items of the order basket
FR20	User should be able to initiate payment for the order	Button	User session should be valid
FR21	User should be able to navigate back to the product listings page	Link	User session should be valid

2.6.8 User Module - Make Payment

Users can use the Payment Gateway integrated into the website using API.

Table 2.7: **Requirements - User Module - Make Payment**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR22	User should be able to provide cardholder's email address in the Stripe Payment form	Input Text box String	email address field cannot be empty email address should be a valid email address
FR23	User should be able to provide billing & shipping address in the Stripe Payment form	Input text boxes Dropdown list String	Address line 1 field cannot be empty Pincode field cannot be empty Town / City field cannot be empty Country Field cannot be empty
FR24	User should be able to provide card details i.e, the card number, card expiry date and CVV number in the Stripe Payment form	Input boxes datetime widget Number	Card Number field cannot be empty card expiry date cannot be blank or a previous date CVV Number field cannot be blank

FR25	User should be able save input card details to the Stripe server	Checkbox boolean	Card details should be filled out for the checkbox to get enabled
FR26	User should be able to exit the Stripe payment form	Exit form button	User should be navigated to the make payment page

2.6.9 User Module - Invoice

Users can receive invoice copy on their registered email addresses as well on the card-holder‘s email address

Table 2.8: **Requirements - User Module - Invoice**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR27	User should receive order invoice copy cum payment receipt upon a successful order placement	Info Panel	User session should be valid Invoice copy cum payment receipt cannot be empty.
FR28	User should receive a unique order ID in the order invoice cum payment receipt for each new order placed.	Output text String	should be unique

FR29	The Order invoice cum payment receipt should contain details of the products ordered that include the image, name, quantity, price per unit and the total price of the product.	Info Panel Image file, String	Products details in the Invoice copy cum payment receipt cannot be empty.
FR30	The Order invoice cum payment receipt should contain the total quantity of products ordered and the total order value	Output text Integer, Number	the total quantity of products ordered and the total order value in the Invoice copy cum payment receipt cannot be empty.
FR31	User should receive error alerts in case of a failed transaction	Alert Message string	user session should be valid

2.6.10 User Module - Notifications

User can receive notifications regarding sign-in, selected products, payment success and order confirmation.

Table 2.9: **Requirements - User Module - Invoice**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR32	User should receive email notification on successful order placement	Email message delivered to user inbox	Email notification shall be sent to the registered email address of the customer Email notification shall be sent to the email address of the cardholder making the payment

2.6.11 Admin Module

Administrator of the website can manage the product catalog and view order data

2.6.12 Admin Module - View Order data

Administrator user of the website should be able to view the orders being placed on the customer web portal

Table 2.10: **Requirements - Admin Module - View Order Data**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR33	Admin user should be able to view order data i.e, Order ID, Order Amount and the Order timestamp	Output text String	None of the fields can be invalid or empty
FR34	Admin user should be able to navigate to the Manage Products page	link	Admin user session should be valid

2.6.13 Admin Module - Manage Products

Admin can create, edit and delete products maintained in the product catalog. User is presented with the list managed by Admin.

Table 2.11: **Requirements - Admin Module - Manage Products**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR35	Admin user should be able to navigate to the 'Add Product Form' by clicking a link or button	Button	Admin user session should be valid
FR36	Admin user should be able to view the product details such as the image, name, quantity and price of the products in the product catalog	Info table Image file, String	Product details should correspond to the ones maintained at the backend database
FR37	Admin user should be able to delete a product from the product catalog	Button	Line item for the product should get removed from the table view Product should exist in the backend database for it to get removed

FR38	Admin user should be able to update a product of the product catalog by navigating to the 'Update Product Catalog' form	Button	Product details should be fetched from the backend database and should get displayed in the update Product Form
FR39	Admin user should be able to navigate to the Admin main view i.e. 'View Order Data'	link	Admin user session should be valid

2.6.14 Admin Module - Add Product

Table 2.12: **Requirements - Admin Module - Add Products**

Req. ID	**Description**	**UI Element & Datatype**	**Validation**
FR40	Admin user should be able to create a product by adding a product name, product quantity,and image	Input Text boxes File Upload utility String, Image file	Product name should be unique Image file name should be unique Image quantity cannot be zero

FR41	Admin user should be able upload an image for the product being added	File Upload utility Image file	image file cannot be empty. Image filename cannot be empty and should be valid
FR42	A unique product ID should get generated for each new product created	UUID UUID	Product ID should unique
FR43	Admin user should be able logoff, go back to the previous page, directly go to the Manage Product page or the View Order data page using links or buttons	Link	Admin user session should be valid

FR44	Admin user should receive error alerts for invalid inputs	Alert boxes string	Empty space is not allowed Fields that require unique value shall only accept such values Fields not meeting the criteria described in the regex pattern for the field shall be invalidated

2.6.15 Reports

Admin can generate basic report of products purchased on the website.

2.7 Software Engineering Paradigm applied

2.7.1 The Personal Software Process (PSP)

The PSP is a structured software development process that is intended (planned) to help software engineers better understand and improve their performance by tracking their predicted and actual development of code. The PSP was created by Watts Humphrey to apply the underlying principles of the Software Engineering Institute‘s (SEI) Capability Maturity Model (CMM) to the software development practices of a single developer. It claims to give software engineers the process skills necessary to work on a Team software process (TSP) team [3].

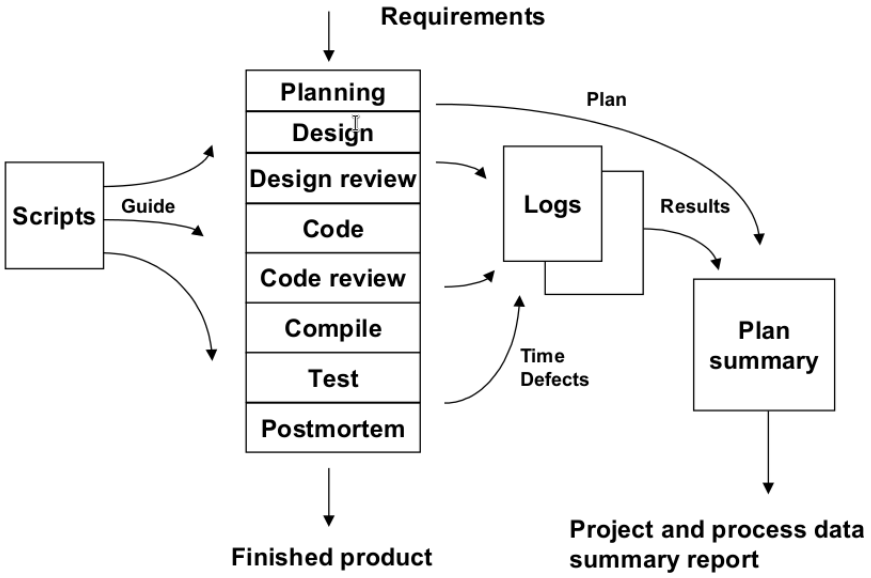


Figure 2.1: PSP Process Flow Diagram

The structure of the PSP process is shown conceptually in the Figure. Starting with a requirements statement, the first step in the PSP process is planning. There is a planning script that guides this work and a plan summary for recording the planning data. While the engineers are following the script to do the work, they record their time and defect data on the time and defect logs. At the end of the job, during the postmortem phase (PM), they summarize the time and defect data from the logs, measure the program size, and enter these data in the plan summary form. When done, they deliver the finished product along with the completed plan summary form.

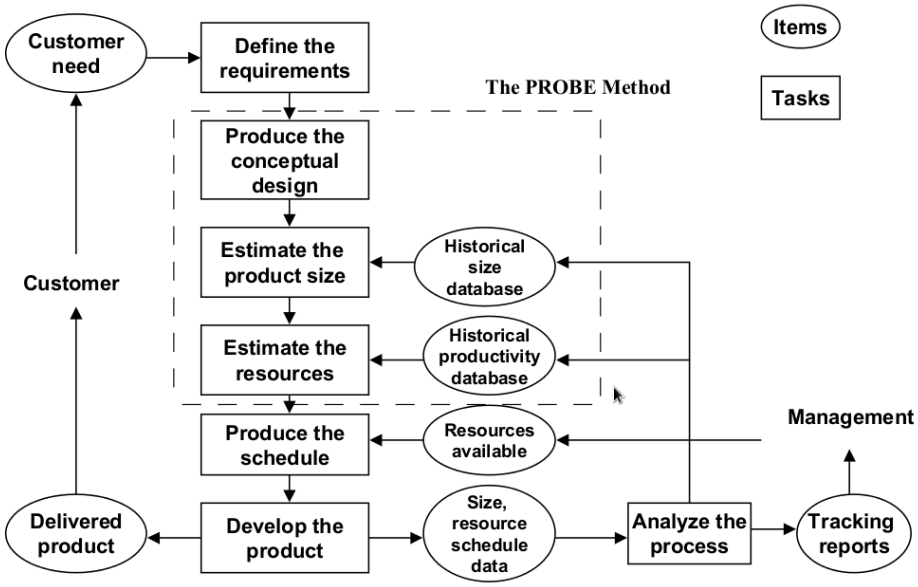


Figure 2.2: PSP Planning Process Diagram

Requirements
Engineers start planning by defining the work that needs to be done in as much detail as possible. If all they have is a one-sentence requirements statement, then that statement must be the basis for the plan

Conceptual Design.
To make an estimate and a plan, engineers first define how the product is to be designed and built. However, since the planning phase is too early to produce a complete product design, engineers produce what is called a conceptual design. This is a first, rough guess at what the product would look like if the engineers had to build it based on what they currently know. Later, during the design phase, the engineers examine design alternatives and produce a complete product design.

2.7.2 Notes on the Go Programming Language Design [4]

The Go programming language was conceived in late 2007 as an answer to some of the problems that were being seen in developing software infrastructure at Google. Today‘s programs may comprise of tens of millions of lines of code, are worked on by hundreds or even thousands of programmers, and are updated literally every day. To make matters worse, build times, even on large compilation clusters, have stretched to many minutes, even hours.

Go was designed and developed to make working in this environment more productive. Besides its better-known aspects such as built-in concurrency and garbage collection, Go’s design considerations include rigorous dependency management, the adaptability of software architecture as systems grow, and robustness across the boundaries between components.

Go is a compiled, concurrent, garbage-collected, statically typed language developed at Google. It is an open source project: Google imports the public repository rather than the other way around. Go is efficient, scalable, and productive.

The goals of the Go project were to eliminate the slowness and clumsiness of software development at Google, and thereby to make the process more productive and scalable. The language was designed by and for people who write, read and debug and maintain large software systems.
The primary considerations in developing the Go programming language were:

- It must work at scale, for large programs with large numbers of dependencies, with large teams of programmers working on them.
- It must be familiar, roughly C-like.The need to get programmers productive quickly in a new language means that the language cannot be too radical.
- It must be modern. C, C++, and to some extent Java are quite old, designed before the advent of multi-core machines, networking, and web application development. There are features of the modern world that are better met by newer approaches, such as built-in concurrency.

Dependencies in Go
Dependencies are defined, syntactically and semantically, by the language. They are explicit, clear, and "computable", which is to say, easy to write tools to analyze.

- language defines that unused dependencies are a compile-time error
- Package invocations are transitive and restricted to the other packages imported by the package thus improving compilation time.
- To make compilation even more efficient, the object file is arranged so the export data is the first thing in the file, so the compiler can stop reading as soon as it reaches the end of that section.
- Another feature of the Go dependency graph is that it has no cycles. The language defines that there can be no circular imports in the graph, and the compiler and linker both check that they do not exist.

Packages

- The design of Go's package system combines some of the properties of libraries, namespaces, and modules into a single construct.
- An important property of Go's package system is that the package path, being in general an arbitrary string, can be co-opted to refer to remote repositories by having it identify the URL of the site serving the repository.

Syntax

- Go was therefore designed with clarity and tooling in mind, and has a clean syntax.
- Unlike C and Java and especially C++, Go can be parsed without type information or a symbol table; there is no type-specific context. The grammar is easy to reason about and therefore tools are easy to write.
- Function syntax is straightforward for simple functions. A method is just a function with a special parameter, its receiver, which can be passed to the function using the standard "dot" notation. Go has first-class functions and closures. Finally, in Go functions can return multiple values.
- One feature missing from Go is that it does not support default function arguments. One mitigating factor for the lack of default arguments is that Go has easy-to-use, type-safe support for variadic functions.

Naming

- Go takes an unusual approach to defining the visibility of an identifier. The case of the initial letter of the identifier determines the visibility. If the initial character is an upper case letter, the identifier is exported (public); otherwise it is not.
- Another simplification is that Go has a very compact scope hierarchy:
    - universe (predeclared identifiers such as int and string)
    - package (all the source files of a package live at the same scope)
    - file (for package import renames only; not very important in practice)
    - function
    - block
- The rules for naming provide an important property for scaling because they guarantee that adding an exported name to a package can never break a client of that package. The naming rules decouple packages, providing scaling, clarity, and robustness.

Semantics

- The semantics of Go statements is generally C-like. That said, Go makes many small changes to C semantics, mostly in the service of robustness. These include:
    - there is no pointer arithmetic
    - there are no implicit numeric conversions
    - array bounds are always checked

- there are no type aliases (after type X int, X and int are distinct types not aliases)
- ++ and – are statements not expressions
- assignment is not an expression
- it is legal (encouraged even) to take the address of a stack variable

- There are some much bigger changes too, stepping far from the traditional C, C++, and even Java models. These include linguistic support for:
  - concurrency
  - garbage collection
  - interface types
  - reflection
  - type switches

Concurrency
Concurrency is important to the modern computing environment with its multicore machines running web servers with multiple clients, what might be called the typical Google program. This kind of software is not especially well served by C++ or Java, which lack sufficient concurrency support at the language level.

Garbage Collection
Go has no explicit memory-freeing operation: the only way allocated memory returns to the pool is through the garbage collector. The language is much easier to use because of garbage collection.

Composition not Inheritance
Go takes an unusual approach to object-oriented programming, allowing methods on any type, not just classes, but without any form of type-based inheritance like sub-classing. This means there is no type hierarchy. Instead, Go has interfaces. In Go an interface is just a set of methods. A type will usually satisfy many interfaces, each corresponding to a subset of its methods. Go encourages composition over inheritance, using simple, often one-method interfaces to define trivial behaviors that serve as clean, comprehensible boundaries between components.

Error
Go does not have an exception facility in the conventional sense, that is, there is no control structure associated with error handling. The key language feature for error handling is a pre-defined interface type called error that represents a value that has an Error method returning a string.

2.8 Data Models & Diagrams

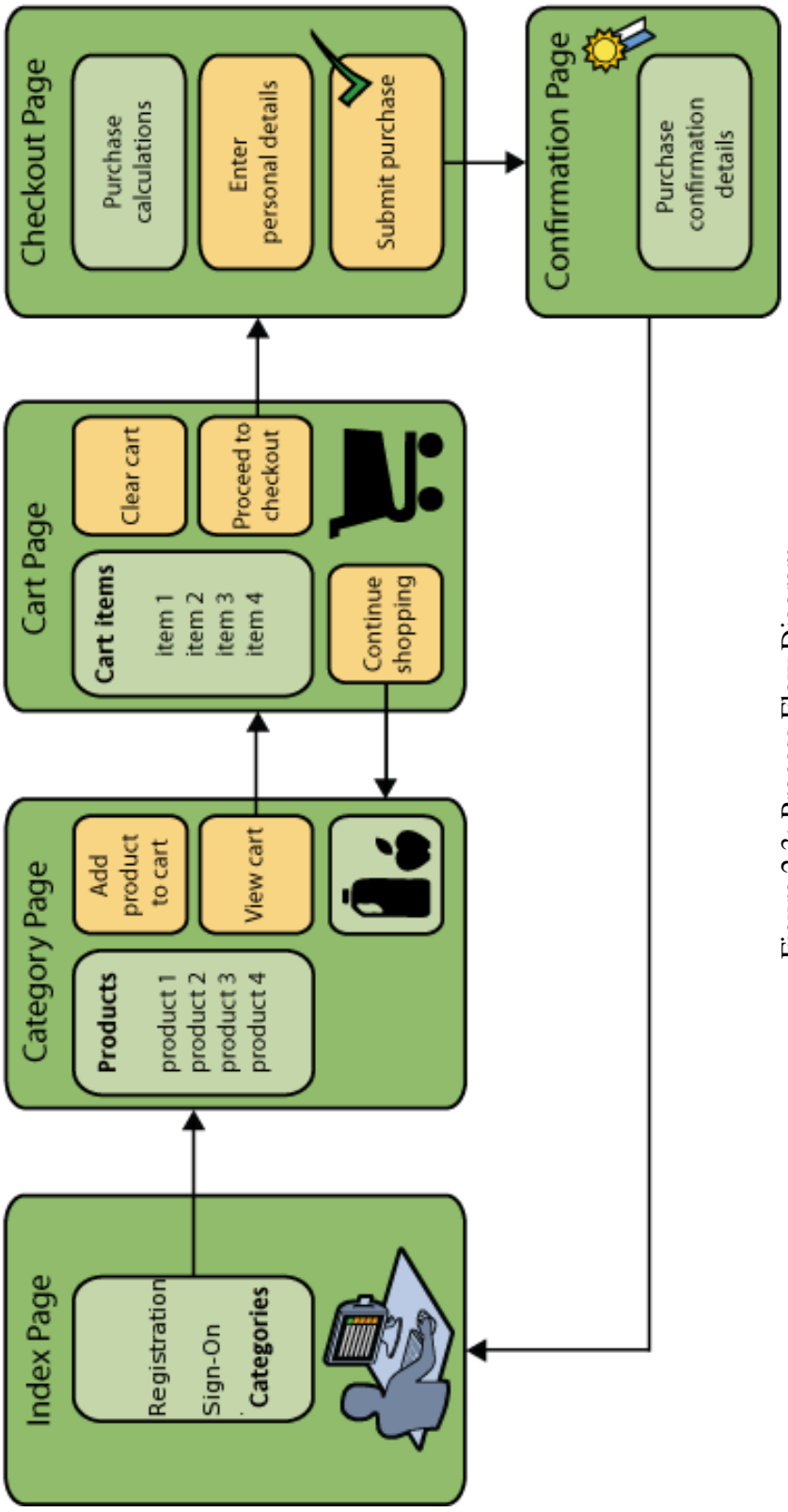


Figure 2.3: Process Flow Diagram

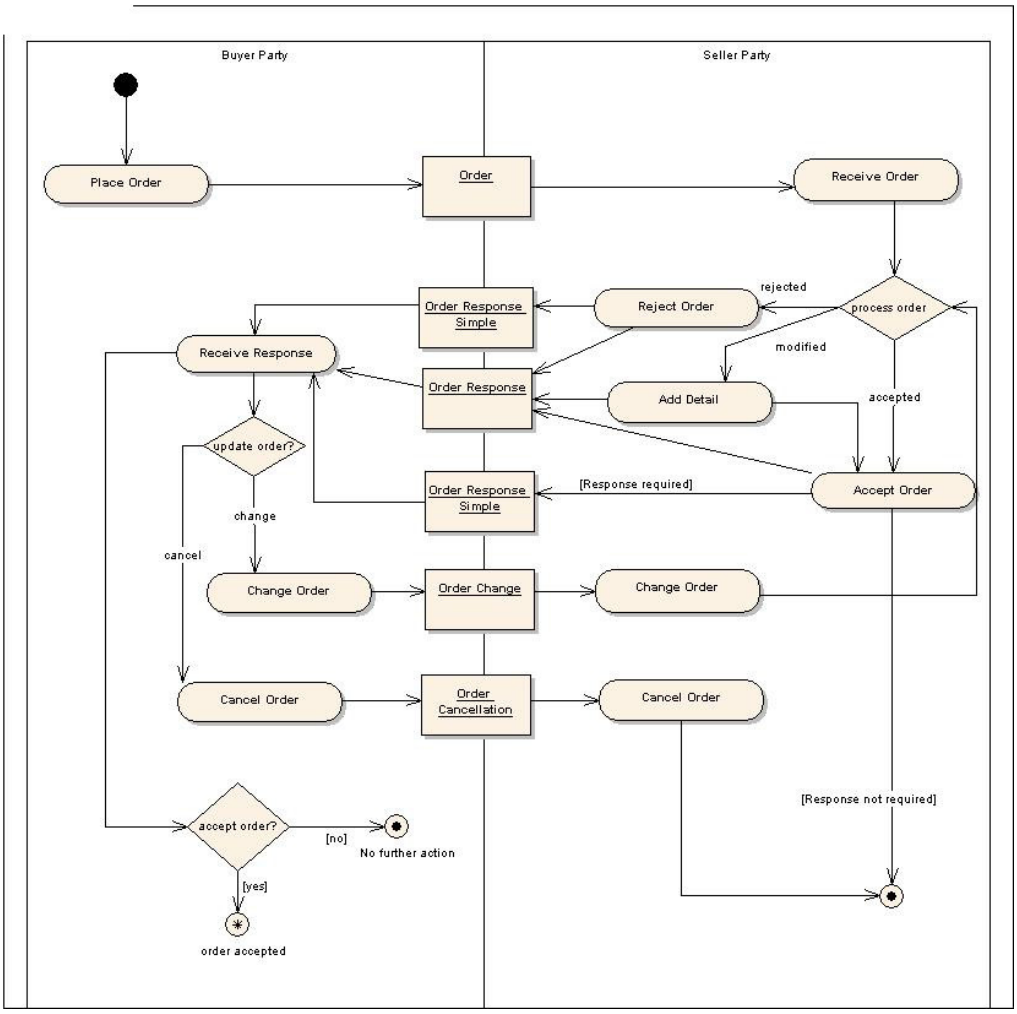


Figure 2.4: Ordering Process Flow Diagram

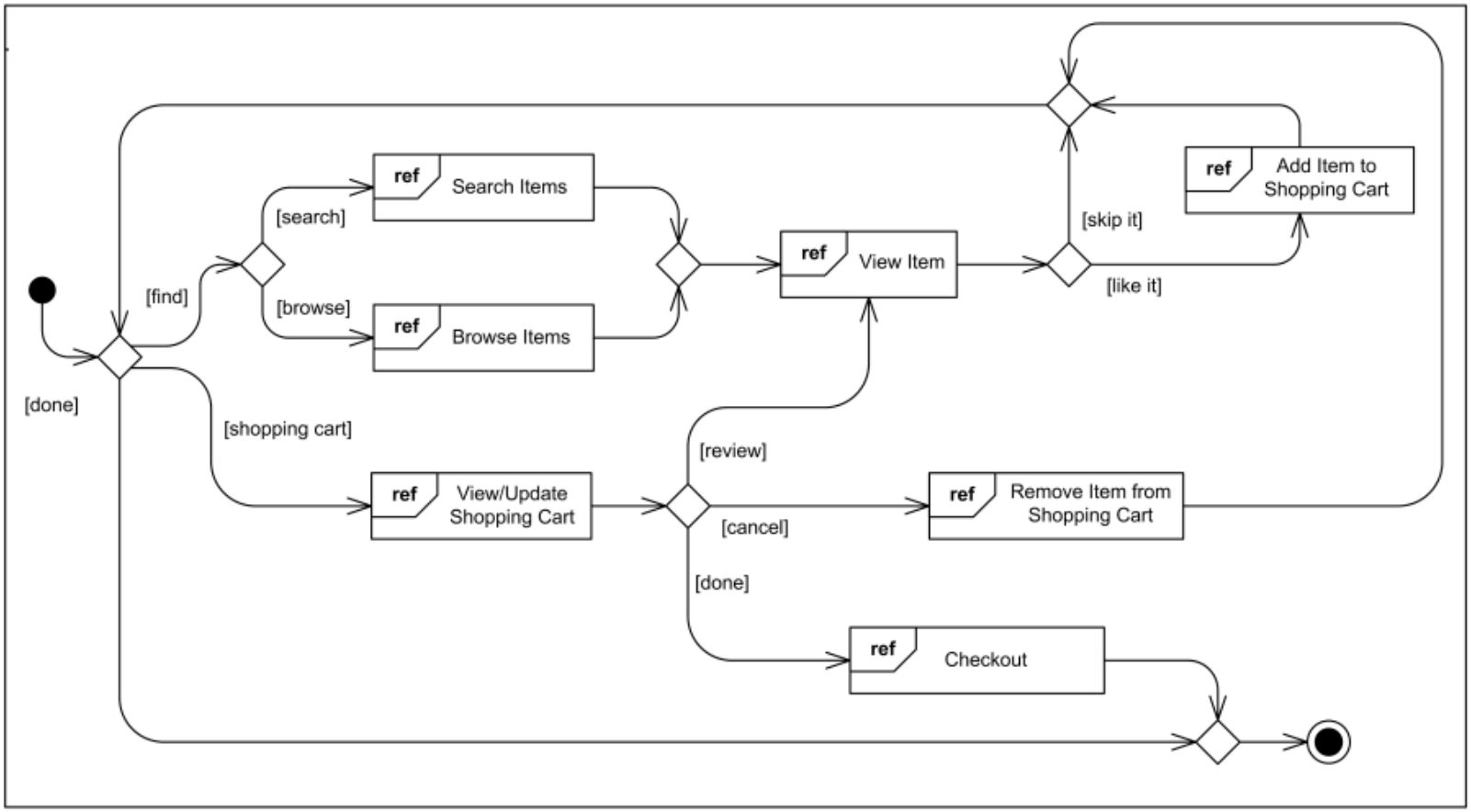


Figure 2.5: Shopping Cart Process Flow Diagram

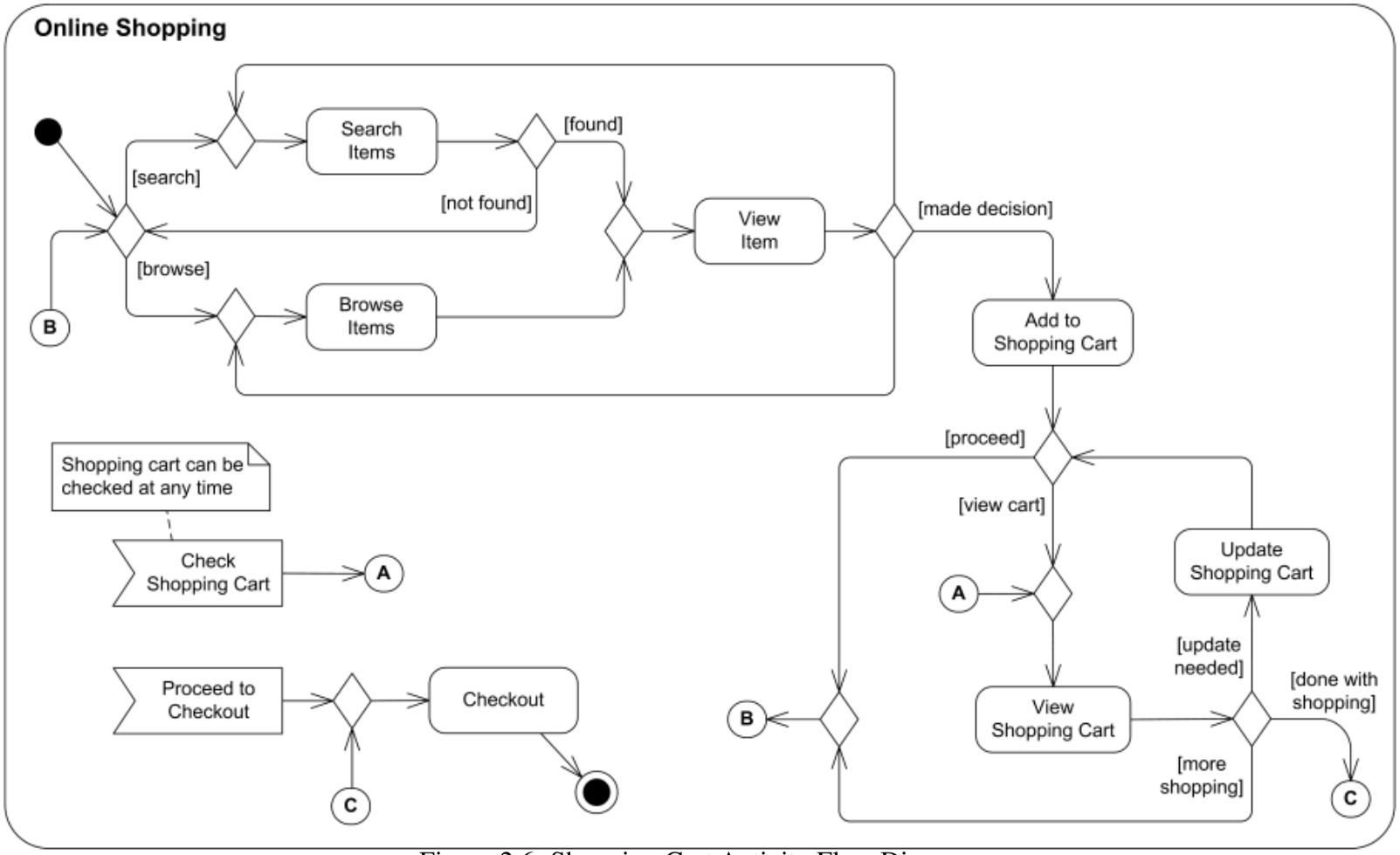


Figure 2.6: Shopping Cart Activity Flow Diagram

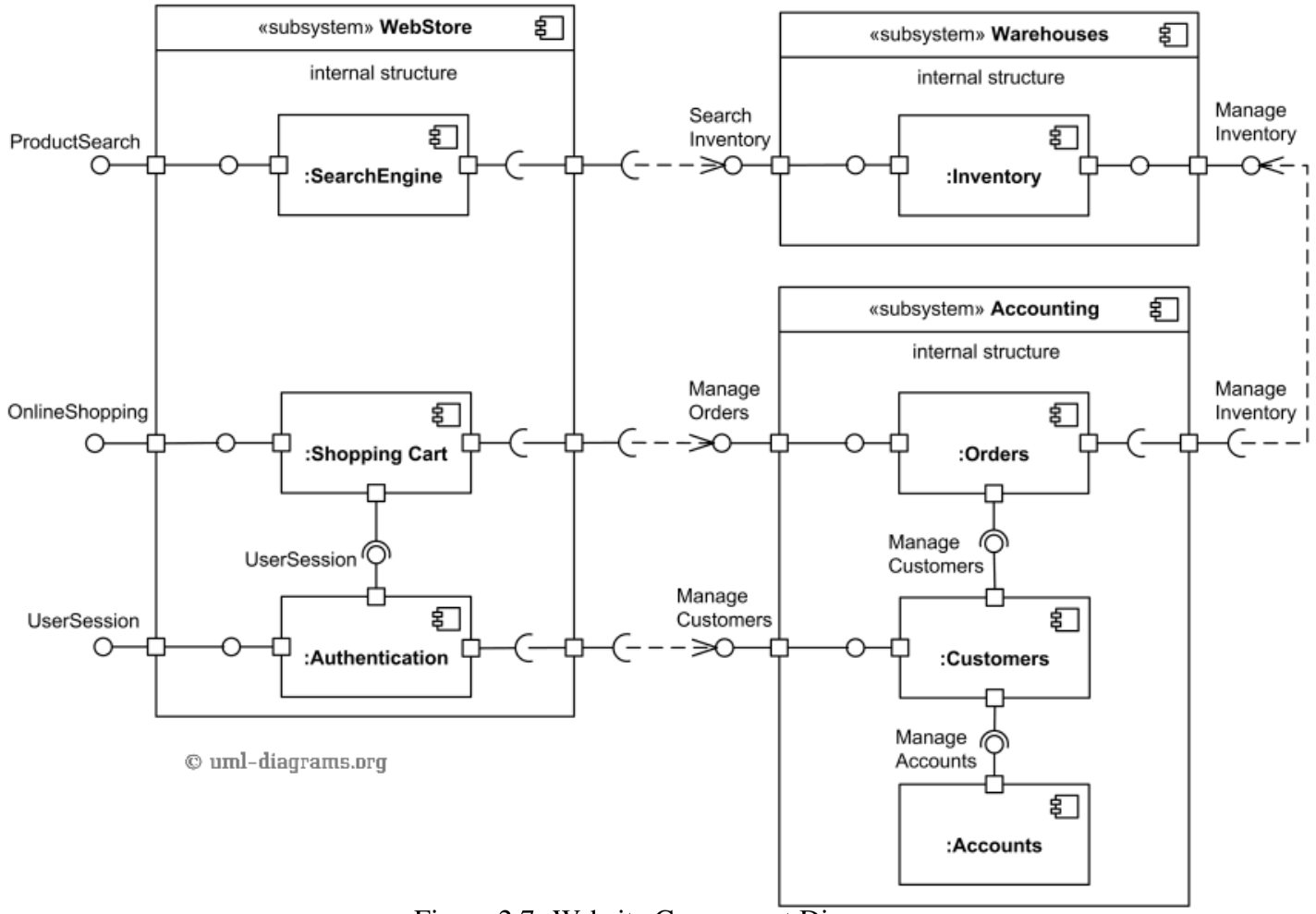


Figure 2.7: Website Component Diagram

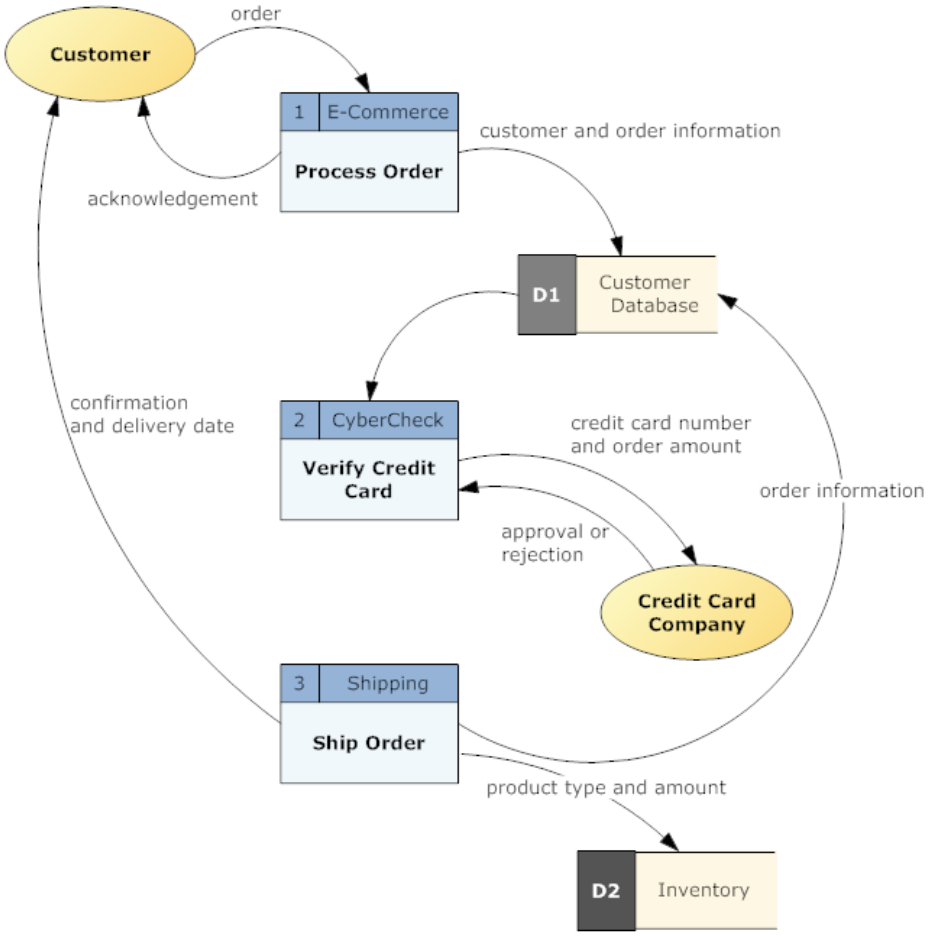


Figure 2.8: Data Flow Diagram

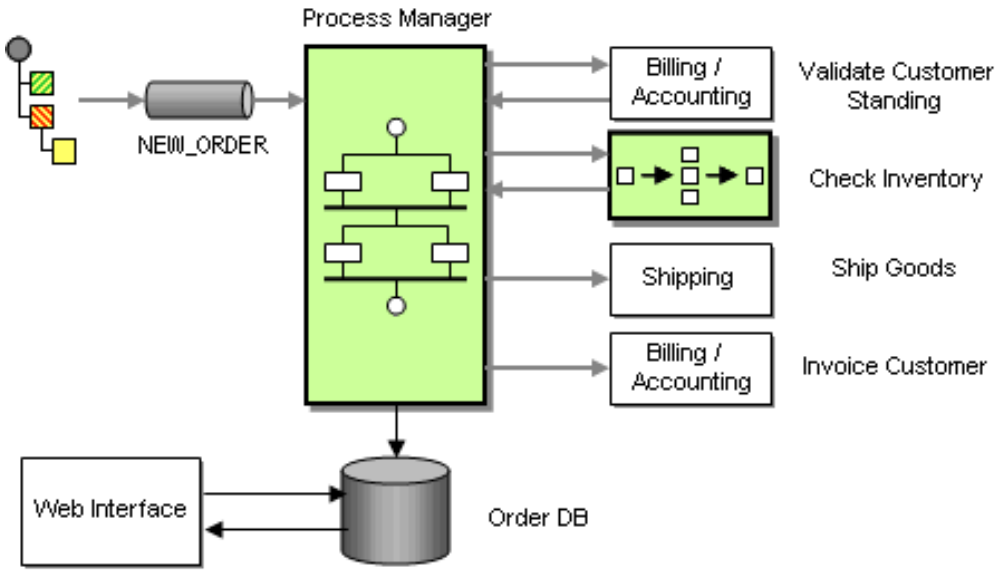


Figure 2.9: Architecture Diagram

3 SYSTEM DESIGN

3.1 Modularization Details

Table 3.1: **Modularization Details**

Sr. No.	**Module**	**Description**
1.0	User Registration & Sign-On	Users can register on the website. Users need to authenticate during login
2.0	Manage Session	Unique session is created for every user after login. Session is destroyed after user logs out
3.0	Shopping Cart	Javascript module for users to add items to shopping cart
4.0	Product Search	Javsacript function for users to be able to search for products using the name string
5.0	Place Order	Triggers order for the selected items in the Cart. A unique Order ID is generated and details are saved to the Database
6.0	Stripe Payment Gateway	API to receive payments for Orders
7.0	Email Notification for Order Confirmation	Automated email notification on successful order and successful payment for the order

8.0	Manage Products	Utility to add products to the database that includes the File Upload Handler
9.0	File Upload Handler	Utility to add Product image when uploading details of a new product
10.0	View Orders	Admin user can query and view details of Placed orders
11.0	View Specific Order	Admin user can query and view details of a specific order

3.2 Database Design

Database design is the process of producing a detailed data model of database. This data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database.

Table	Children	Parents	Columns
delprod			1
orders			10
products			5
sqlite_stat1			3
users			6
5 Tables			**25**

Figure 3.1: List of Database Tables

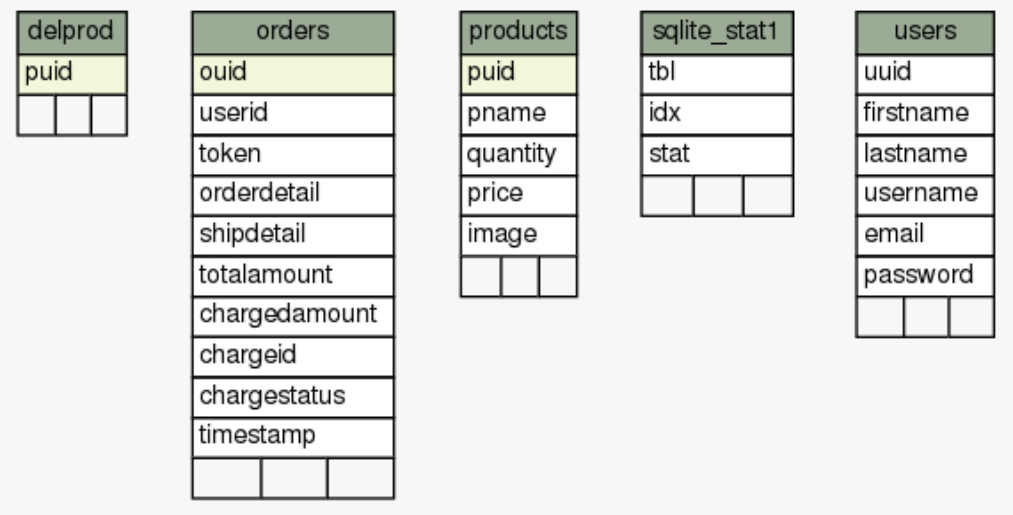


Figure 3.2: Utility Tables with data Elements

users.sqlite3 contains 25 columns - click on heading to sort:

Table	Column	Type	Size	Nulls	Auto	Default
delprod	puid	text	2000000000,10			
orders	chargedamount	integer	2000000000,10			
orders	chargeid	text	2000000000,10			
orders	chargestatus	text	2000000000,10			
orders	orderdetail	blob	2000000000,10			
orders	ouid	text	2000000000,10			
orders	shipdetail	blob	2000000000,10			
orders	timestamp	text	2000000000,10	√		null
orders	token	text	2000000000,10			
orders	totalamount	text	2000000000,10			
orders	userid	text	2000000000,10			
products	image	text	2000000000,10	√		null
products	pname	text	2000000000,10			
products	price	text	2000000000,10			
products	puid	text	2000000000,10			
products	quantity	text	2000000000,10			
sqlite_stat1	idx		2000000000,10	√		null
sqlite_stat1	stat		2000000000,10	√		null
sqlite_stat1	tbl		2000000000,10	√		null
users	email	text	2000000000,10			
users	firstname	text	2000000000,10			
users	lastname	text	2000000000,10			
users	password	text	2000000000,10			
users	username	text	2000000000,10			
users	uuid	text	2000000000,10			

Figure 3.3: Data Elements type and size

Table users.sqlite3.orders

☑ Related columns ☑ Constraints ☐ Comments ☑ Legend

Column	Type	Size	Nulls	Auto	Default	Children	Parents
ouid	text	2000000000,10					
userid	text	2000000000,10					
token	text	2000000000,10					
orderdetail	blob	2000000000,10					
shipdetail	blob	2000000000,10					
totalamount	text	2000000000,10					
chargedamount	integer	2000000000,10					
chargeid	text	2000000000,10					
chargestatus	text	2000000000,10					
timestamp	text	2000000000,10	√		null		

Analyzed at Wed Feb 22 10:45 IST 2017

Indexes:

Column(s)	Type	Sort	Constraint Name
ouid	Must be unique	Asc	sqlite_autoindex_orders_1

Figure 3.4: Order table

Table users.sqlite3.products

☑ Related columns ☑ Constraints ☐ Comments ☑ Legend

Column	Type	Size	Nulls	Auto	Default	Children	Parents
puid	text	2000000000,10					
pname	text	2000000000,10					
quantity	text	2000000000,10					
price	text	2000000000,10					
image	text	2000000000,10	√		null		

Analyzed at Wed Feb 22 10:45 IST 2017

Indexes:

Column(s)	Type	Sort	Constraint Name
puid	Must be unique	Asc	sqlite_autoindex_products_1

Figure 3.5: Product table

Table users.sqlite3.users

☑ Related columns ☑ Constraints ☐ Comments ☑ Legend

Column	Type	Size	Nulls	Auto	Default	Children	Parents
uuid	text	2000000000,10					
firstname	text	2000000000,10					
lastname	text	2000000000,10					
username	text	2000000000,10					
email	text	2000000000,10					
password	text	2000000000,10					

Analyzed at Wed Feb 22 10:45 IST 2017

Figure 3.6: User table

Column	Type	Size	Nulls	Auto	Default	Children	Parents
puid	text	2000000000,10					

Analyzed at Tue Mar 07 23:07 IST 2017

Indexes:

Column(s)	Type	Sort
puid	Must be unique	Asc

Figure 3.7: Delete Product table

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.
JSON is a format that encodes objects in a string. Serialization means to convert an object into that string, and deserialization is its inverse operation. When transmitting data or storing them in a file, the data are required to be byte strings.

The ‘orderdetails‘ and ‘shipdetails‘fields are maintained as JSON data (Blob type) in the ‘Orders‘ table of the database

Figure 3.8: Orderdetails Field JSON Structure

Figure 3.9: Shipdetails Field JSON Structure

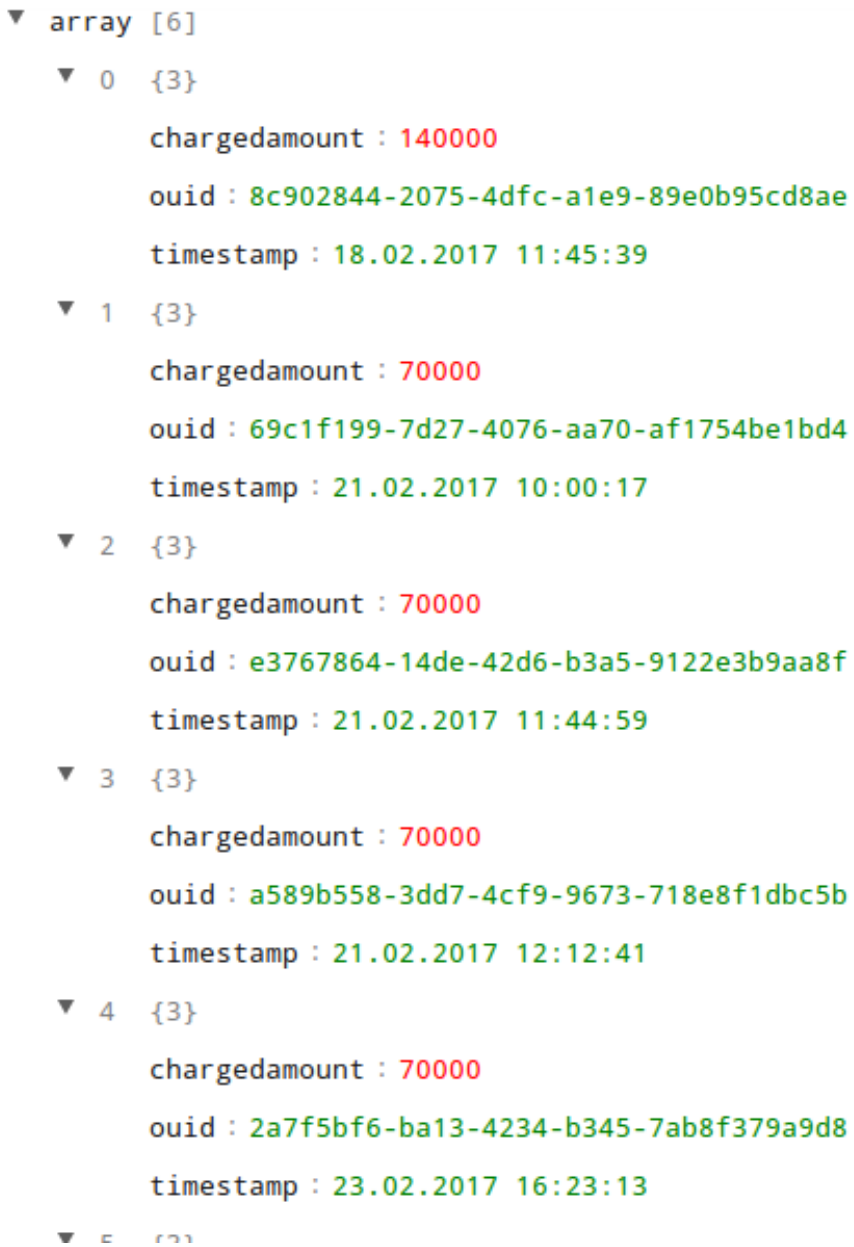


Figure 3.10: JSON Array View Order

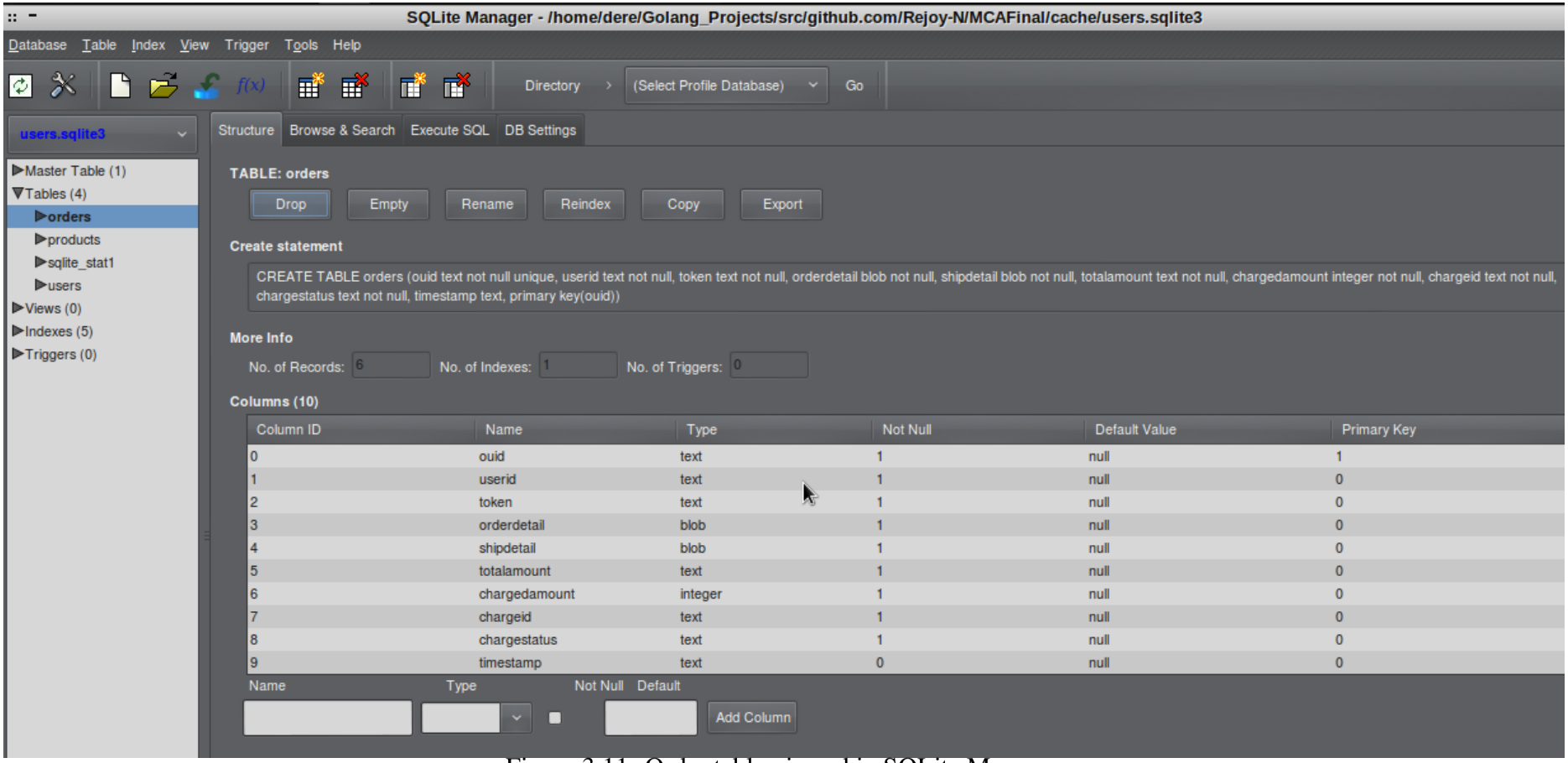


Figure 3.11: Order table viewed in SQLite Manager

With local storage, web applications can store data locally within the user‘s browser.Local storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

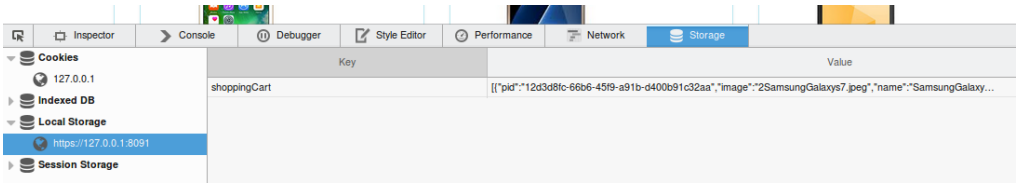


Figure 3.12: HTML Local Storage

An HTTP cookie (also called web cookie, Internet cookie, browser cookie or simply cookie) is a small piece of data sent from a website and stored on the user‘s computer by the user‘s web browser while the user is browsing. Cookies were designed to be a reliable mechanism for websites to remember stateful information (as in this case items added in the shopping cart in an online store)

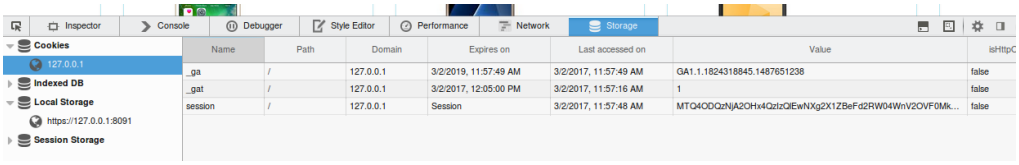


Figure 3.13: Browser Session Cookies

3.3 User Interface Design

User interface design (UI) or user interface engineering is the design of user interfaces for machines and software, such as computers, home appliances, mobile devices, and other electronic devices, with the focus on maximizing usability and the user experience. The goal of user interface design is to make the user‘s interaction as simple and efficient as possible, in terms of accomplishing user goals (user-centered design). [5]

A good e-commerce site should present the following factors to users for better usability

- Simple navigation from home page to information and order links for specific products
- Obvious shopping links or buttons
- Effective categorical Organization of products
- Easy scanning and selecting items on list
- Consistent layout of product information
- Minimal or effective security notification or messages
- Knowing when an item was saved or not saved in the shopping cart

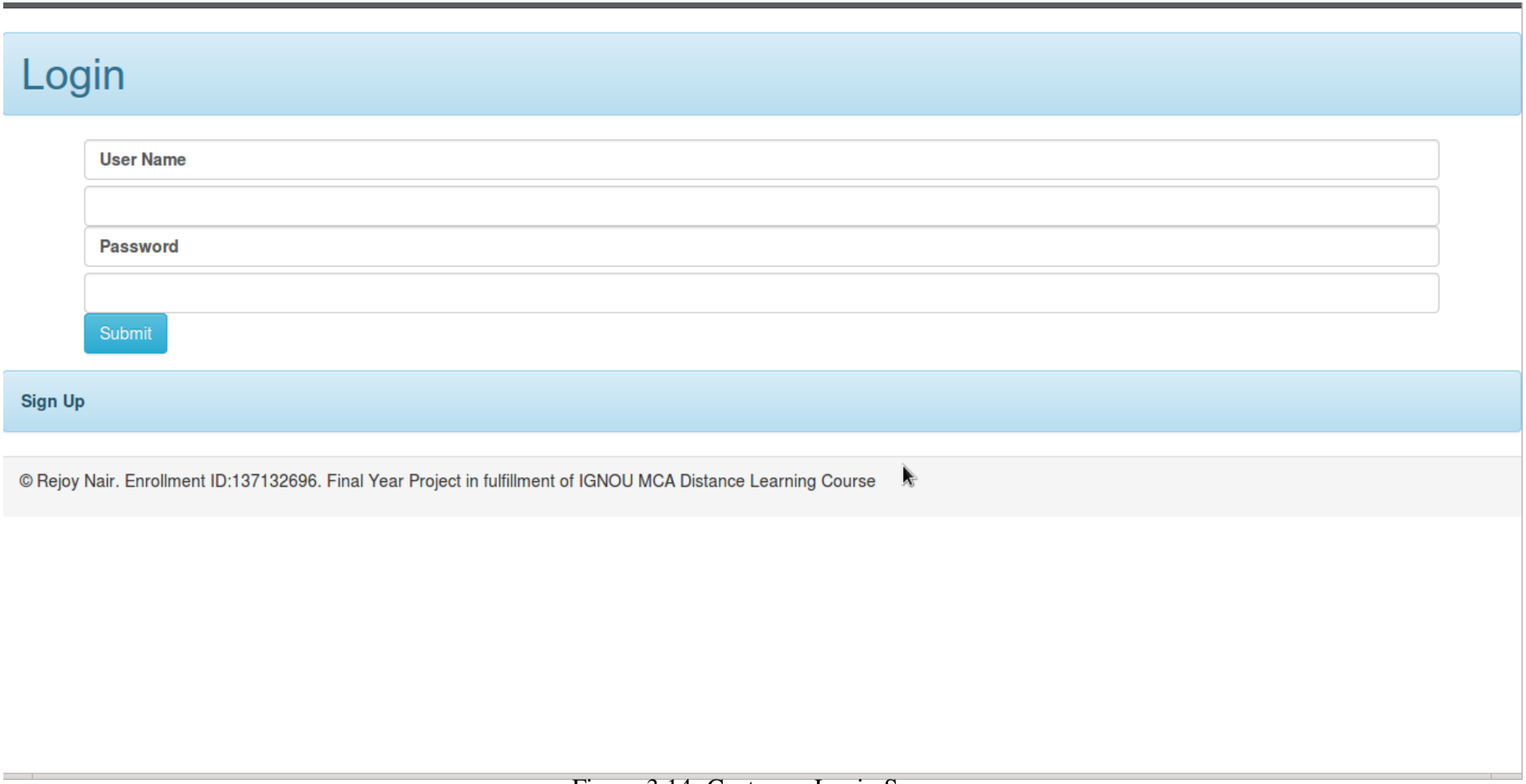


Figure 3.14: Customer Login Screen

Table 3.2: **User Login**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Username Input Field	Textbox to accept registered username. Email ID or any other unique attribute shall not be accepted
2.0	Password Input Field	Textbox to accept password associated with the Username
3.0	Submit Button	To post the Username and password entered by the user to the server
4.0	Alert	To notify User of incorrect or empty Username and / or password or an expired User session

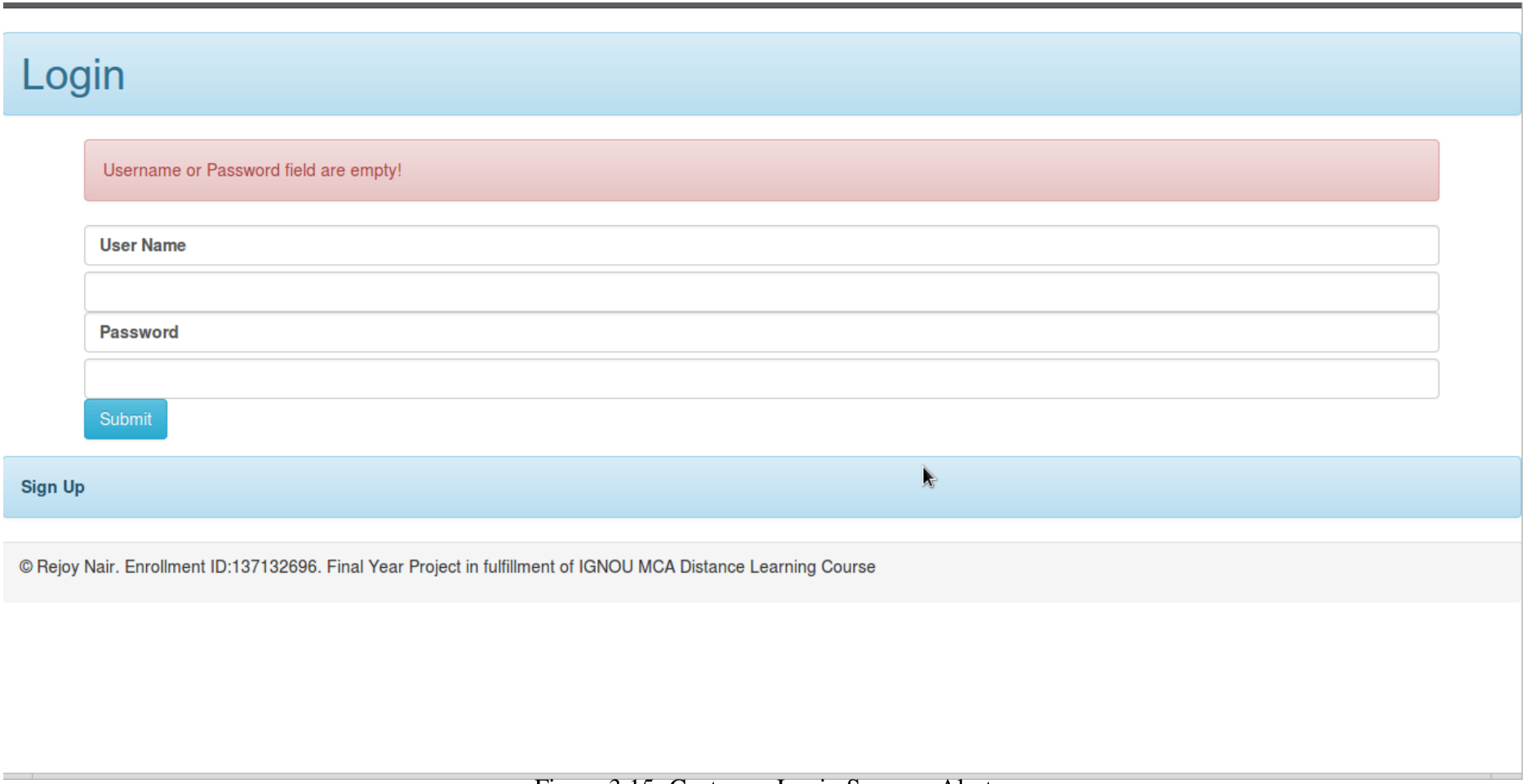


Figure 3.15: Customer Login Screen - Alert

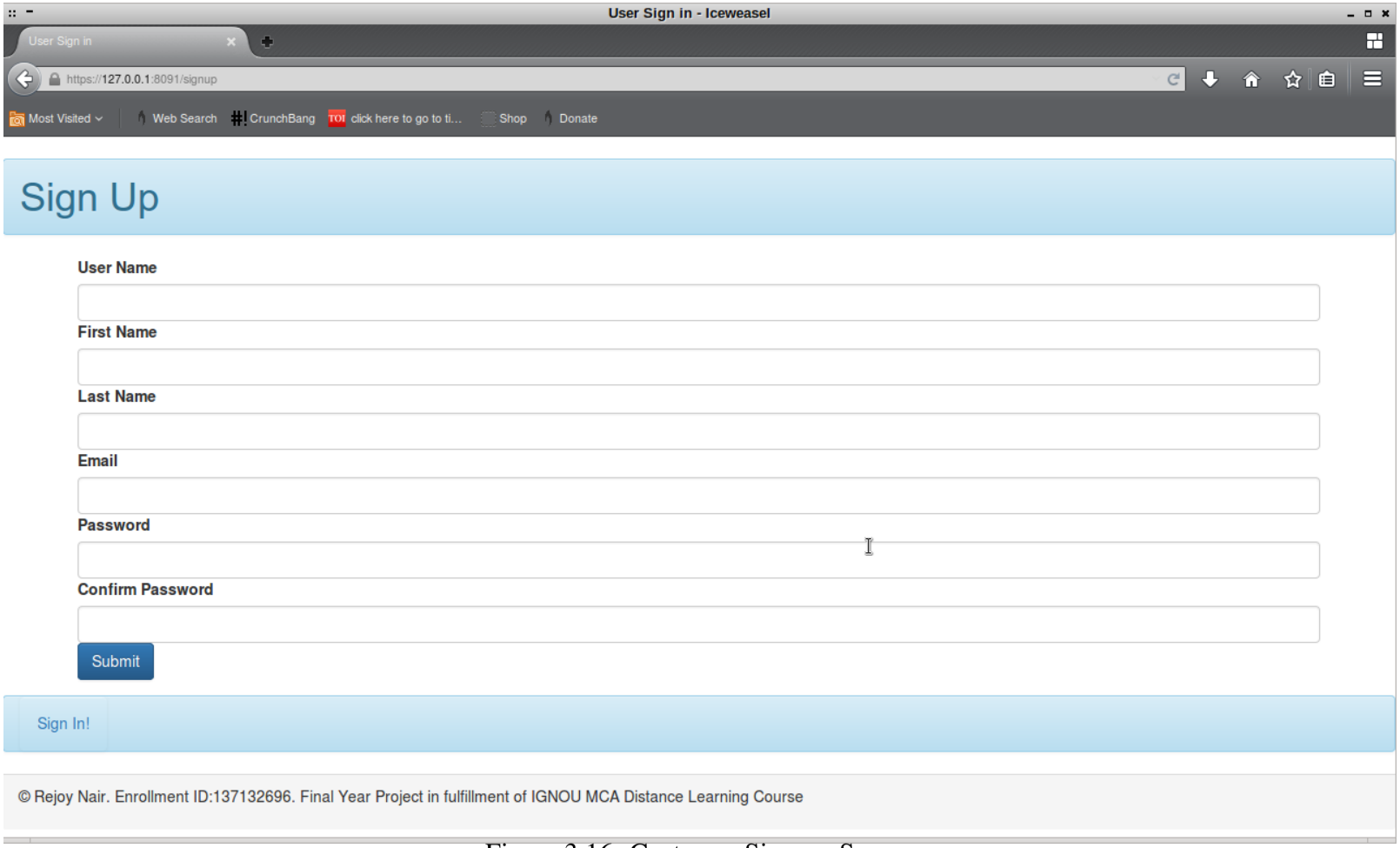


Figure 3.16: Customer Sign-up Screen

Table 3.3: **Customer Signup Screen**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Username Input Field	Textbox to accept username that the user wishes to register with. Username will have to be unique
2.0	First Name Input Field	Textbox to accept first name that the user wishes to register with.
3.0	Last Name Input Field	Textbox to accept last name that the user wishes to register with.
4.0	Email Input Field	Textbox to accept email address that the user wishes to register with.
5.0	Password Input Field	Textbox to create a password for the username that the user wishes to register with.
6.0	Confirm Password Input Field	Textbox to reconfirm the entered password at the time of registration
7.0	Submit Button	To post the details contained in the sign-up form as entered by the user to the server
8.0	Alert	To notify User of incorrect or empty or invalid input entries for each of the fields of the sign-up form

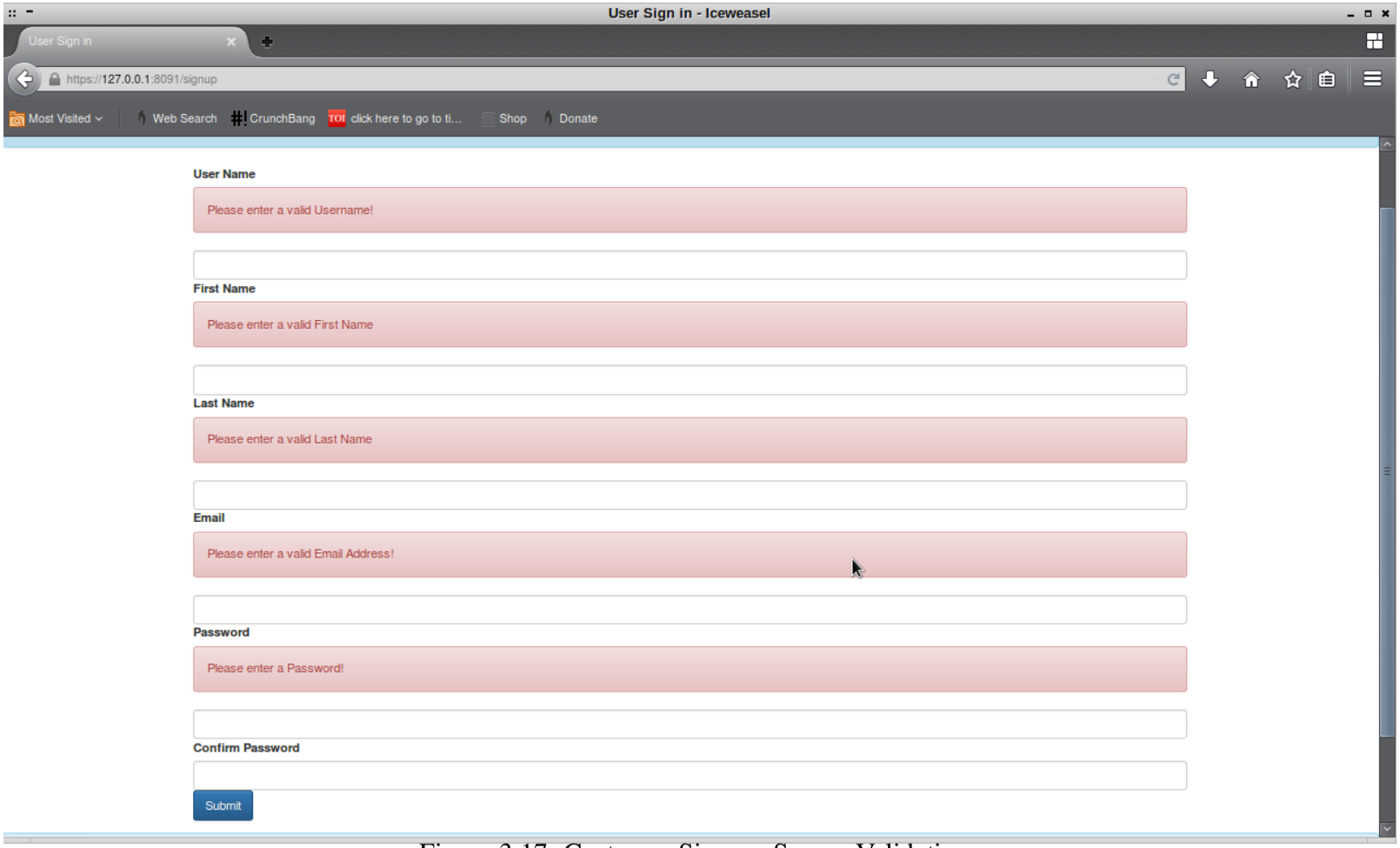


Figure 3.17: Customer Sign-up Screen Validations

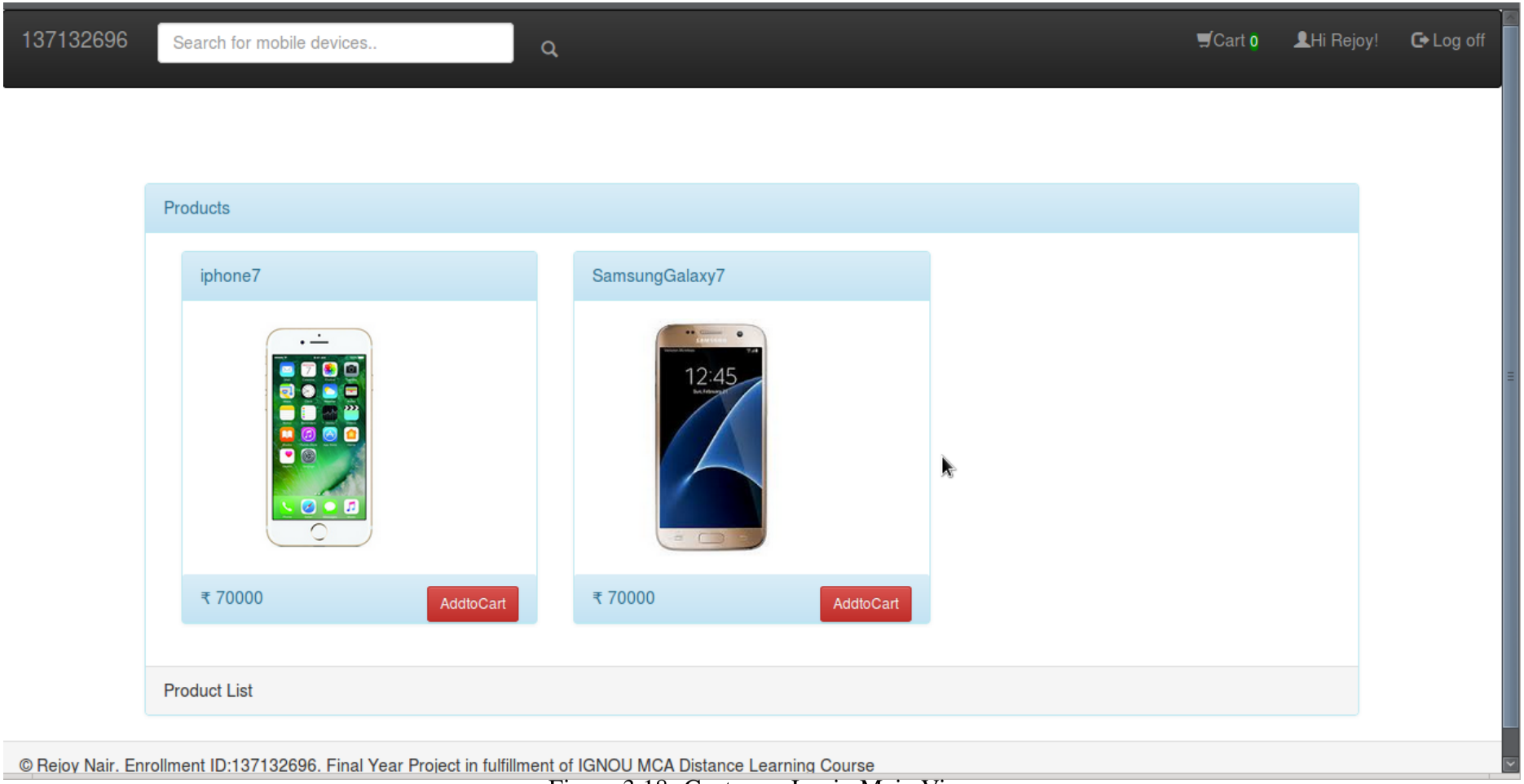


Figure 3.18: Customer Login Main View

Table 3.4: **Customer Login Main View**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Product Info Panel	Panel that contains description of the product such as name and price along with its image
2.0	Product Info Panel – Add to Cart Button	''Add to Cart'' button on the product info panel that allows customers to add products to the cart
3.0	Banner	Banner to place the brand logo. Currently contains the enrollment ID 137132696
4.0	Search Box	Search box to filter products based on Product description
5.0	Shopping Cart	Icon which shows the count of the products added to the shopping cart. Clicking on the shopping cart displays the dropdown that contains the list of the products added to the cart along with the image, price, quantity and total amount per item
6.0	Shopping Cart– Quantity	Number input type that can be increased or decreased. Min Value is 1

7.0	Shopping Cart-Button	‘‘-’’ button to remove a single unit of the item from the total quantity. ‘‘+’’ to add a single unit to the existing quantity. ‘‘X’’ to completely remove the item from the shopping cart list
8.0	Alert	To notify User of a product being added to the shopping cart
9.0	User Profile	Link to the user Profile Page. By defaults shows the user first name.
10.0	Logout	Allows the user to logout from the web portal

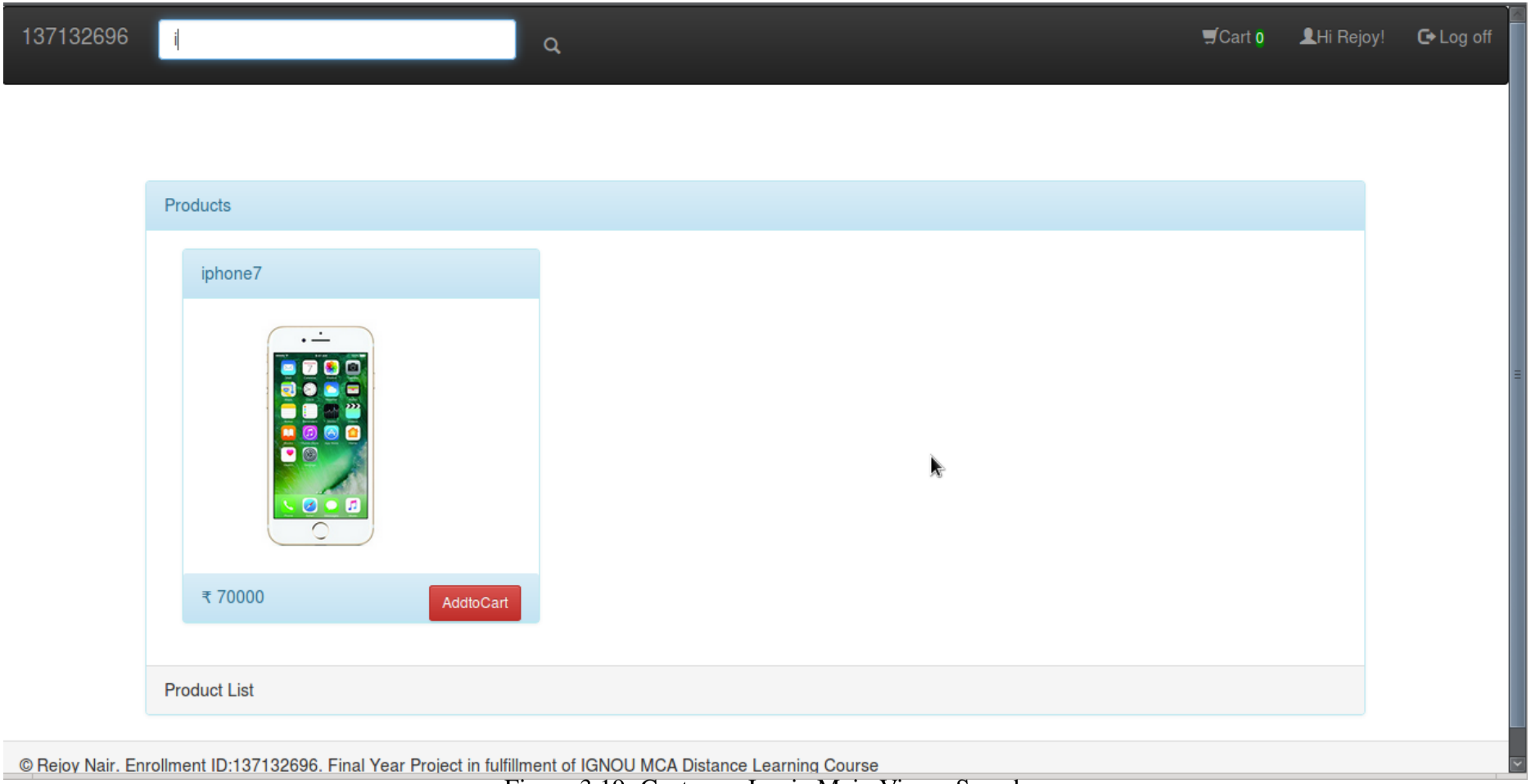


Figure 3.19: Customer Login Main View - Search

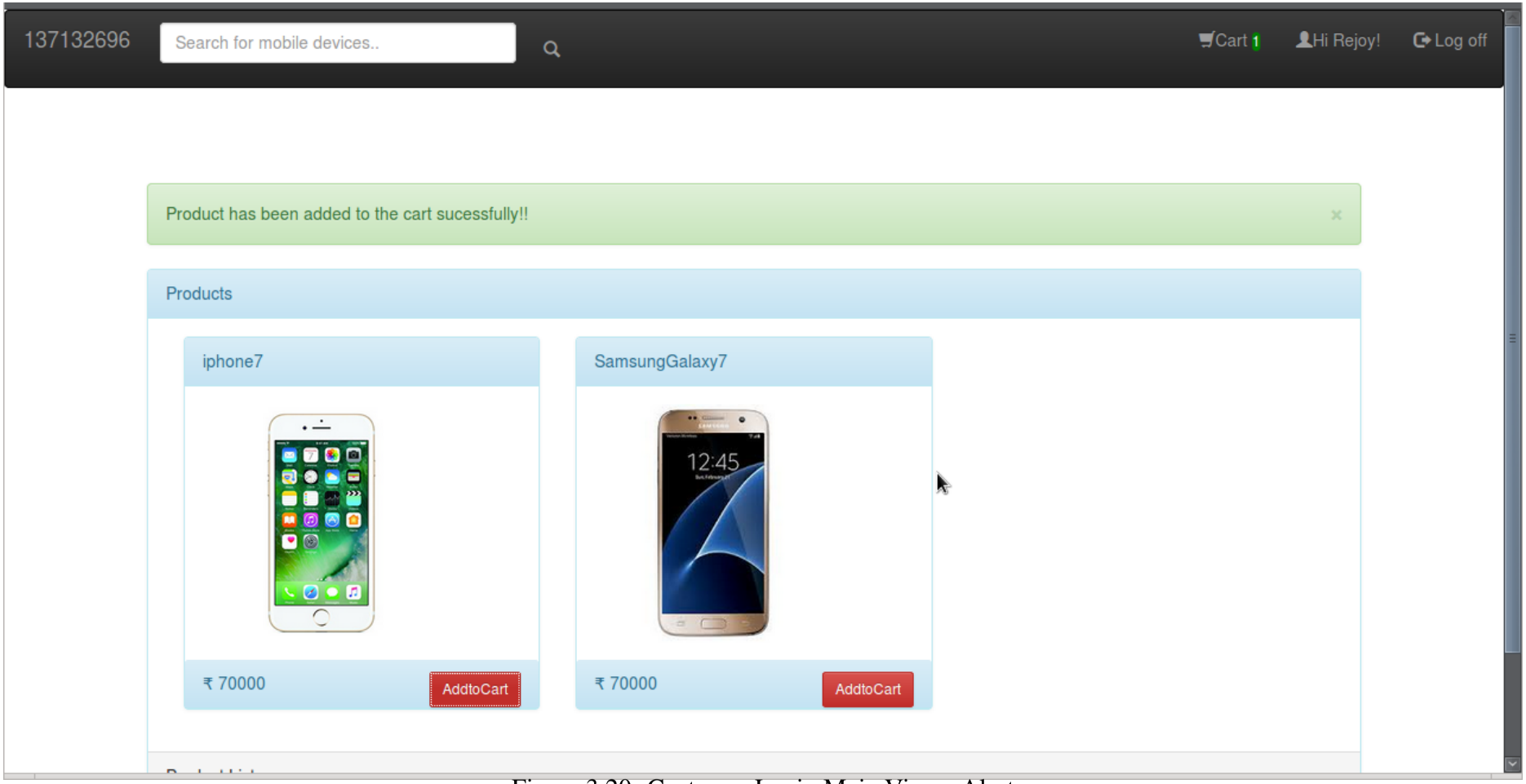


Figure 3.20: Customer Login Main View - Alert

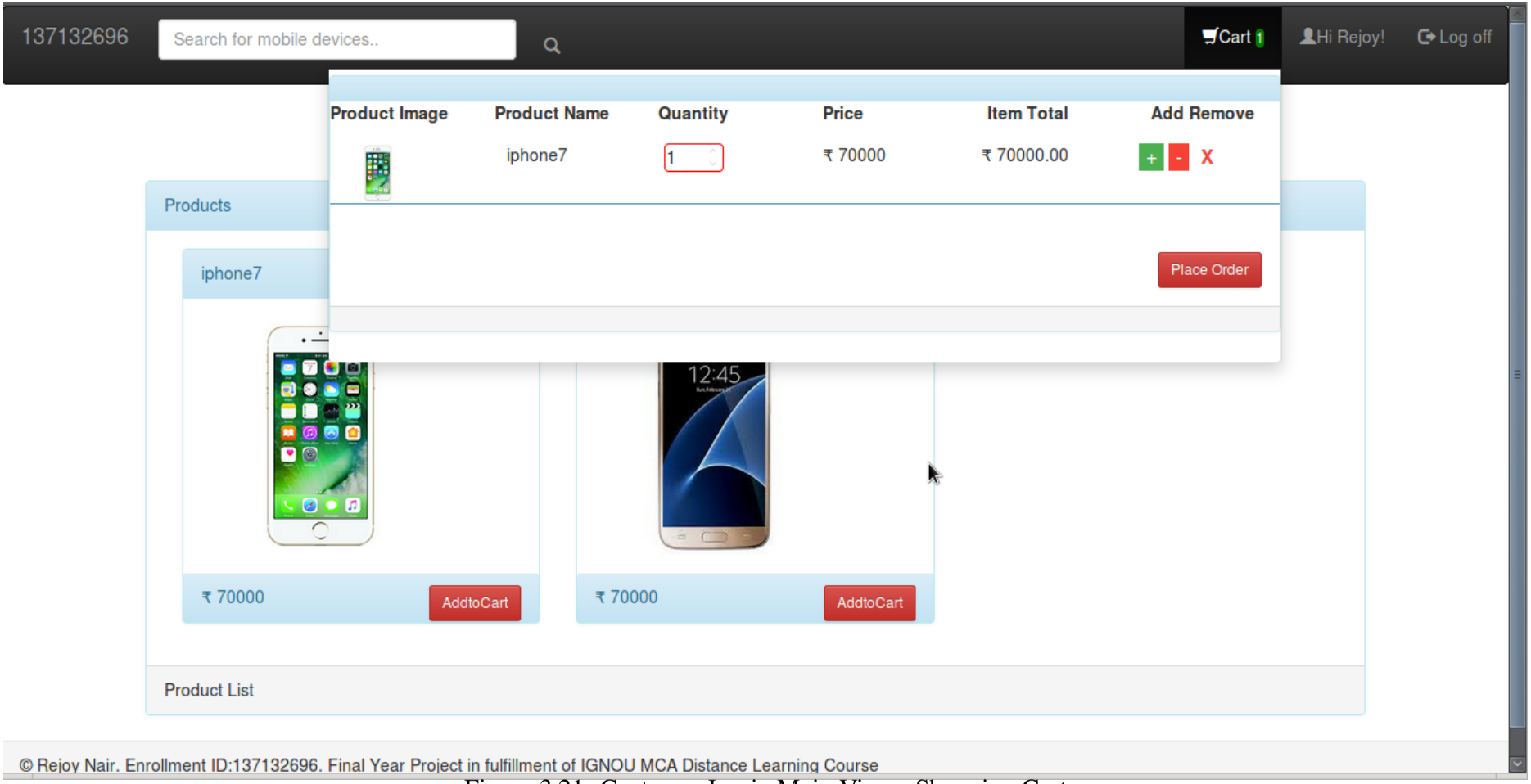


Figure 3.21: Customer Login Main View - Shopping Cart

Table 3.5: **Customer Login Main View**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Product Image	Image of the Product added to the shopping cart
2.0	Product Name	Name of the Product added to the shopping cart
3.0	Quantity	Input number box to increase or decrease the quantity of the product added to the shopping cart
4.0	Price	Price of the product added to the shopping cart
5.0	Item Total	Total cost per line item
6.0	``–'' button	Button to remove a single unit of the product added to the shopping cart
7.0	``+'' button	Button to increase the quantity by a single unit of the product already added to the shopping cart
8.0	``X''	Button to completely remove the product added to the shopping cart
9.0	Place Order	Button to place an order with the items added to the shopping cart

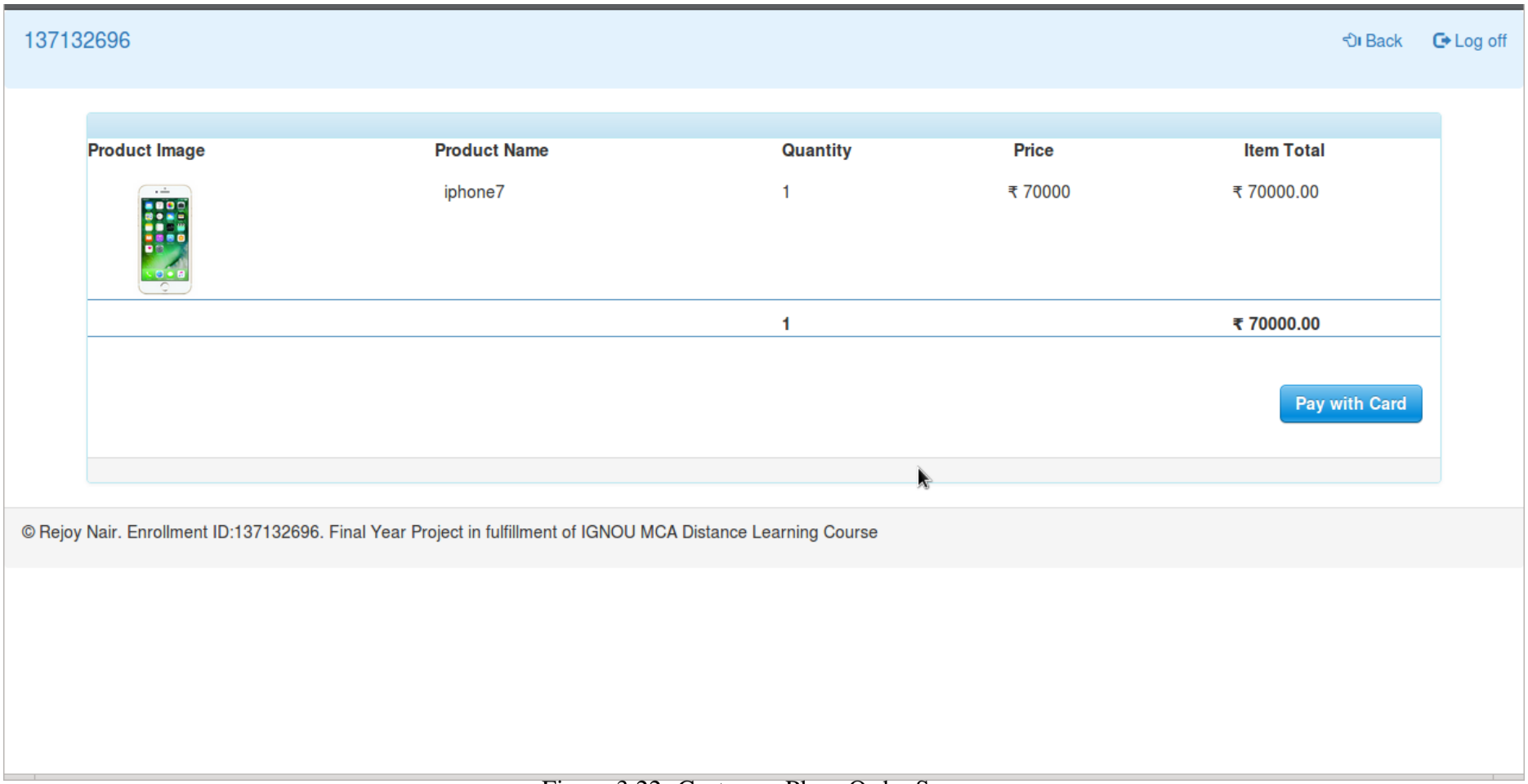


Figure 3.22: Customer Place Order Screen

Table 3.6: **Customer Place Order Screen**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Product Image	Image of the Product added to the shopping cart
2.0	Product Name	Name of the Product added to the shopping cart
3.0	Quantity	Input number box to increase or decrease the quantity of the product added to the shopping cart
4.0	Price	Price of the product added to the shopping cart
5.0	Item Total	Total cost per line item
6.0	Total Count	Total count of products added to shopping cart
7.0	Grand Total	Total Order value
8.0	Pay with Card	Button to call the Stripe's Payment Gateway API
9.0	Back Button	Button to go back to the previous page i.e. the main view
10.0	Logoff Button	Button to logoff from the customer web portal

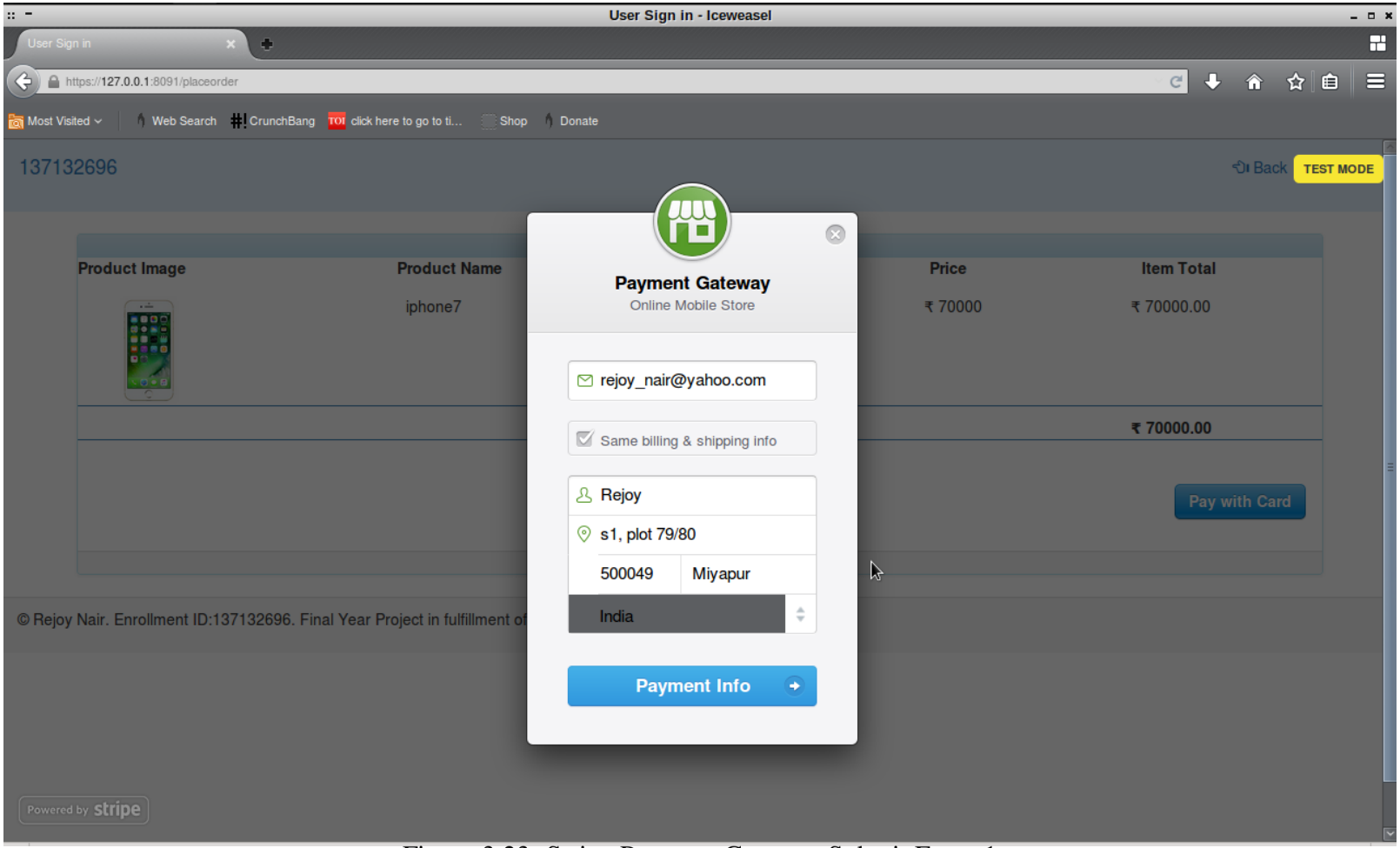


Figure 3.23: Stripe Payment Gateway Submit Form-1

Table 3.7: **Stripe Payment Gateway Submit Form-1**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Label	Label on top of the form
2.0	Email Address	Email Address of the card Holder. This can be different from the registered email address of the customer
3.0	Checkbox – Billing and Shipping info	Checkbox to indicate if the billing and shipping addresses are one and the same. Default is checked.
4.0	Name of the Card Holder	Text box to enter the name of the card holder
5.0	Address Line 1	Text box to enter the address line of the card holder
6.0	Pincode	Text box to enter the pincode corresponding to the address of the card holder
7.0	Town / City Name	Town / city name for the entered pincode. This field is auto-populated based on the entered pincode.
8.0	Payment Info	Button to navigate to the next section of the form where the card information shall be sought
9.0	Grand Total	Total Order value
10.0	Pay with Card	Button to call the Stripe's Payment Gateway API

11.0	Card Number	Text box to enter the card number of the card holder
12.0	Card Expiry date	Calendar widget to enter the card number of the card holder
13.0	Card CVV Number	Input box to enter the CVV number associated with the Card Number of the card holder
14.0	‘Remember me‘ Checkbox	Checkbox to save the cardholder and the card details on Stripe‘s server
15.0	Pay	Button to post card holder information and card details to Stripe‘s server

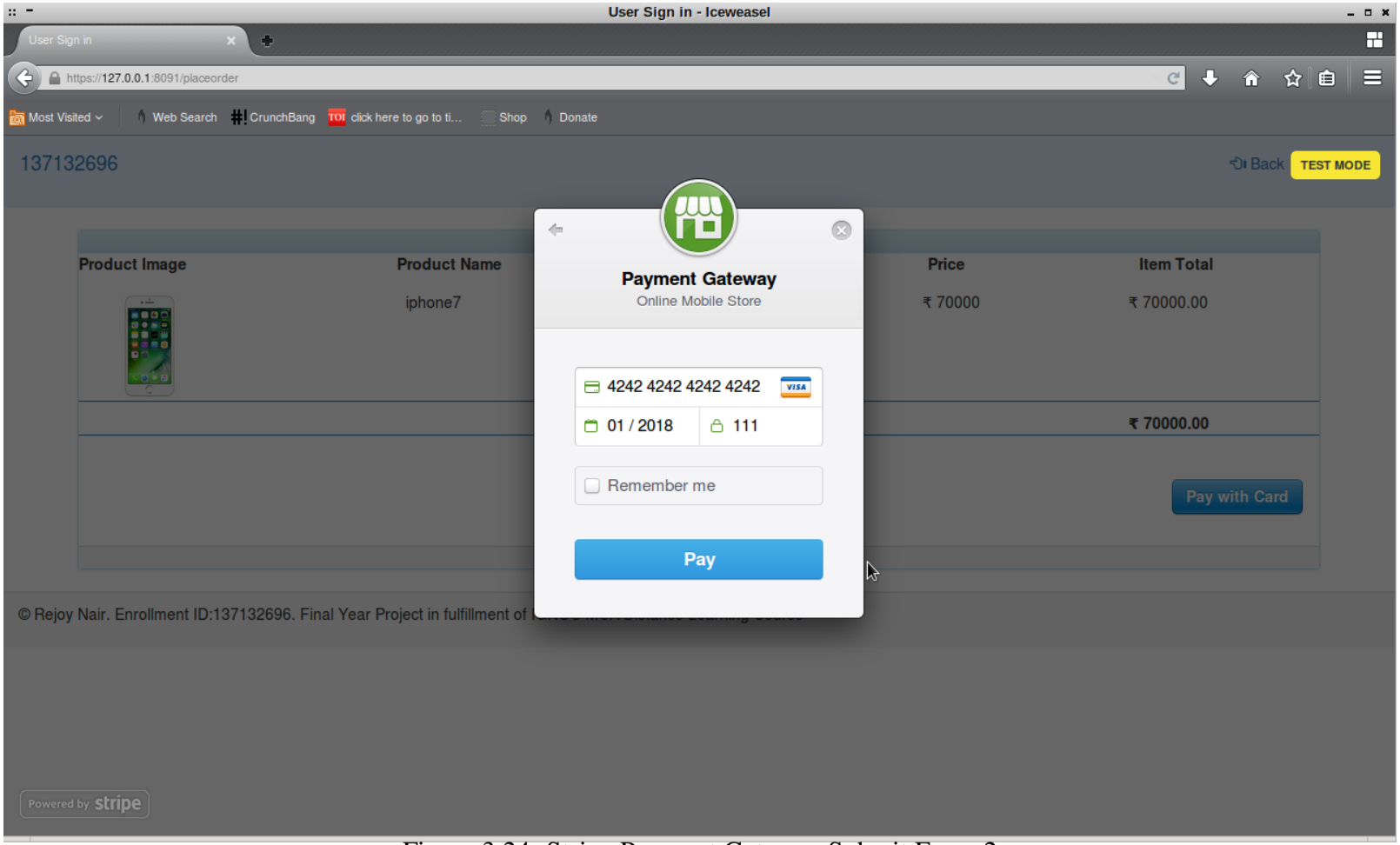


Figure 3.24: Stripe Payment Gateway Submit Form-2

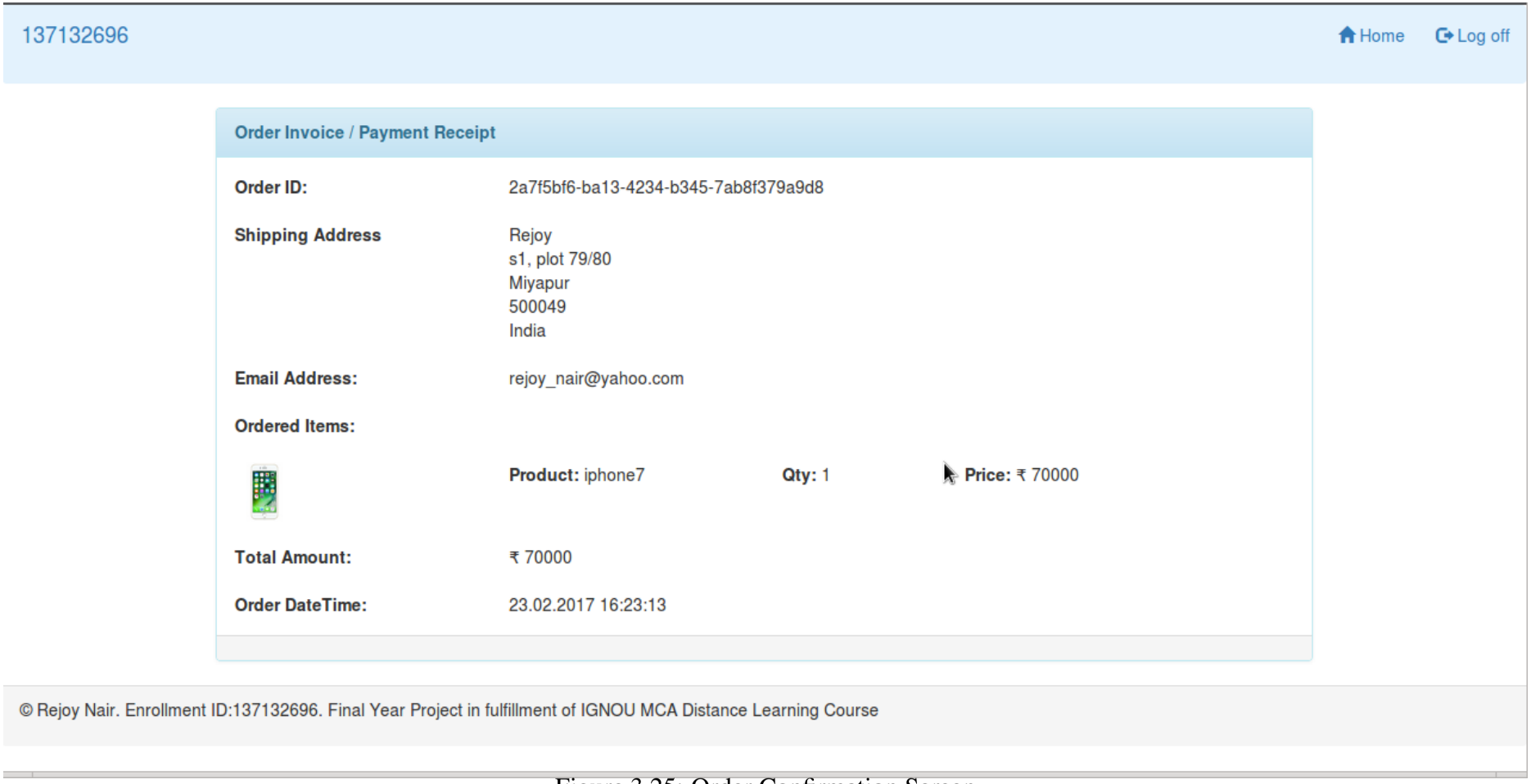


Figure 3.25: Order Confirmation Screen

Table 3.8: **Order Invoice / Payment Receipt**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Order ID	Output text displaying the Order ID field value of the Order Invoice cum payment receipt of the Order placed by the customer
2.0	Shipping Address	Output text displaying the Shipping Address field value of the Order Invoice cum payment receipt of the Order placed by the customer
3.0	Email Address	Output text displaying the Cardholder's Email Address field value of the Order Invoice cum payment receipt of the Order placed by the customer
4.0	Ordered Items - Product Image	Image of the Ordered Product corresponding to the Order Invoice cum payment receipt of the Order placed by the customer
5.0	Ordered Items - Product Name	Output text displaying the Name of the Ordered Product corresponding to the Order Invoice cum payment receipt of the Order placed by the customer

6.0	Ordered Items – Ordered Quantity	Output text displaying the Quantity of the Ordered Product corresponding to the Order Invoice cum payment receipt of the Order placed by the customer
7.0	Ordered Items – Amount	Output text displaying the Per Unit Cost of the Ordered Product corresponding to the Order Invoice cum payment receipt of the Order placed by the customer
8.0	Total Amount	Output text displaying the Total Order Amount corresponding to the Order Invoice cum payment receipt of the Order placed by the customer
9.0	Order DateTime	Output text displaying the local date-time corresponding to the Order Invoice cum payment receipt of the Order placed by the customer
10.0	Home Button	Button to navigate the customer back to the main page of the customer web portal
11.0	Logoff Button	Button to logoff from the customer web portal

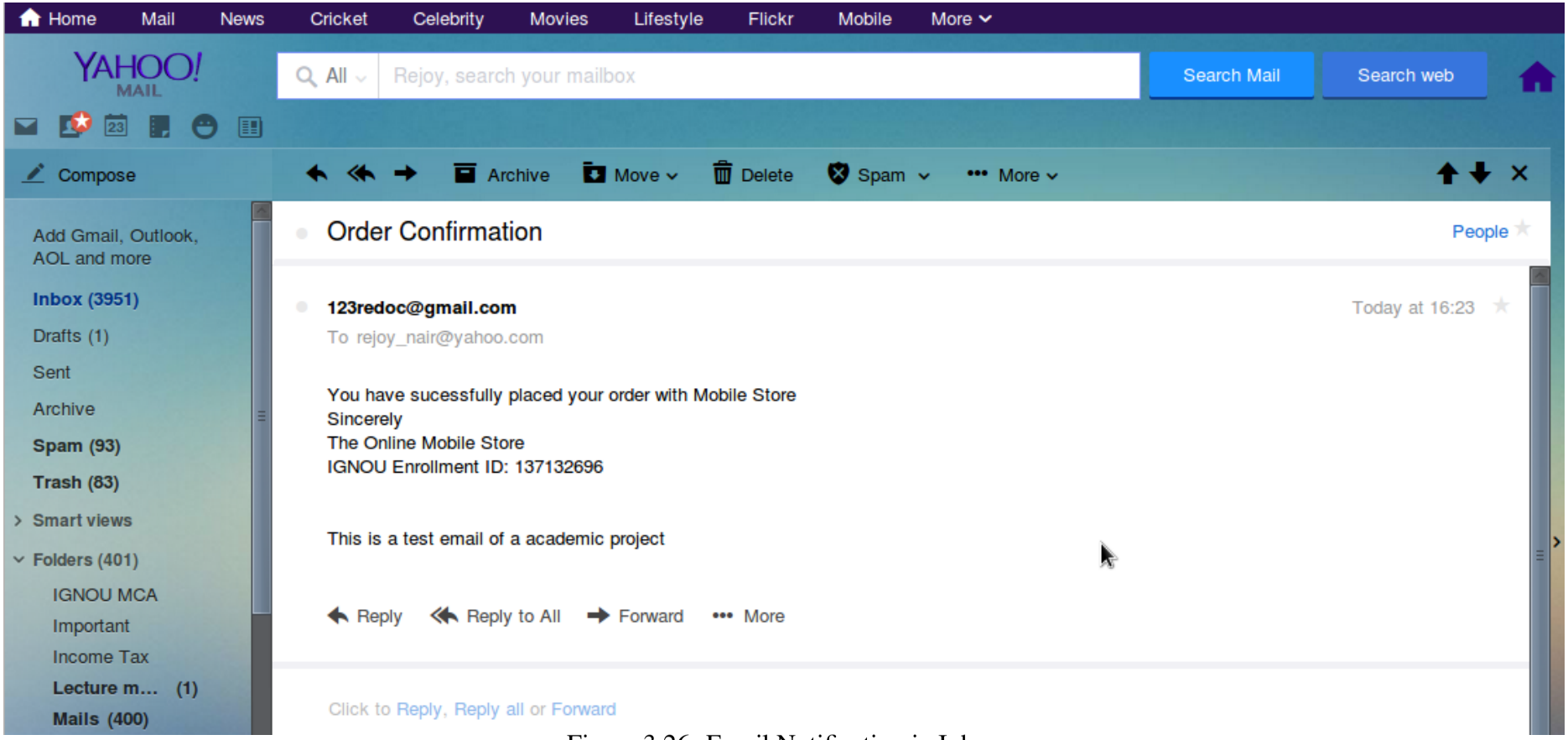


Figure 3.26: Email Notification in Inbox

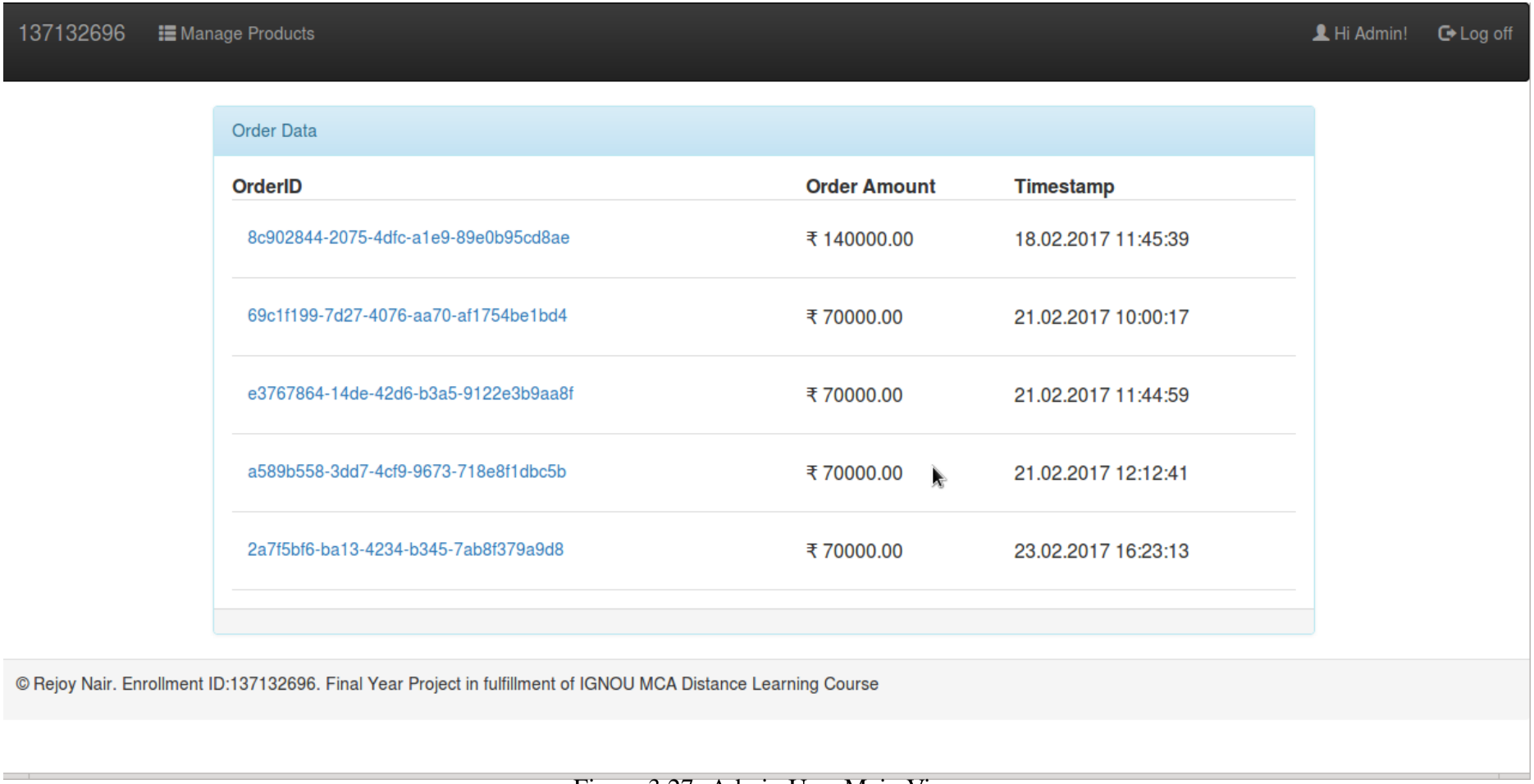


Figure 3.27: Admin User Main View

Table 3.9: **Admin Main View - Order data**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Banner	Banner to place the brand logo. Currently contains the enrollment ID 137132696
2.0	Manage Product	Link to navigate the Admin user to the Manage Products Page
3.0	User Profile icon	Displays the text 'Admin' indicating Admin user login
4.0	Log-off	Link to log-off the Admin User from the Admin User portal
5.0	Order Data – Order ID	Output text displaying the Order ID of the successfully placed orders displayed in the order data table
6.0	Order Data – Order Amount	Output text displaying the total amount corresponding to each order ID of the successfully placed orders displayed in the order data table
7.0	Order Data – Timestamp	Output text displaying the timestamp corresponding to each order ID of the successfully placed orders displayed in the order data table

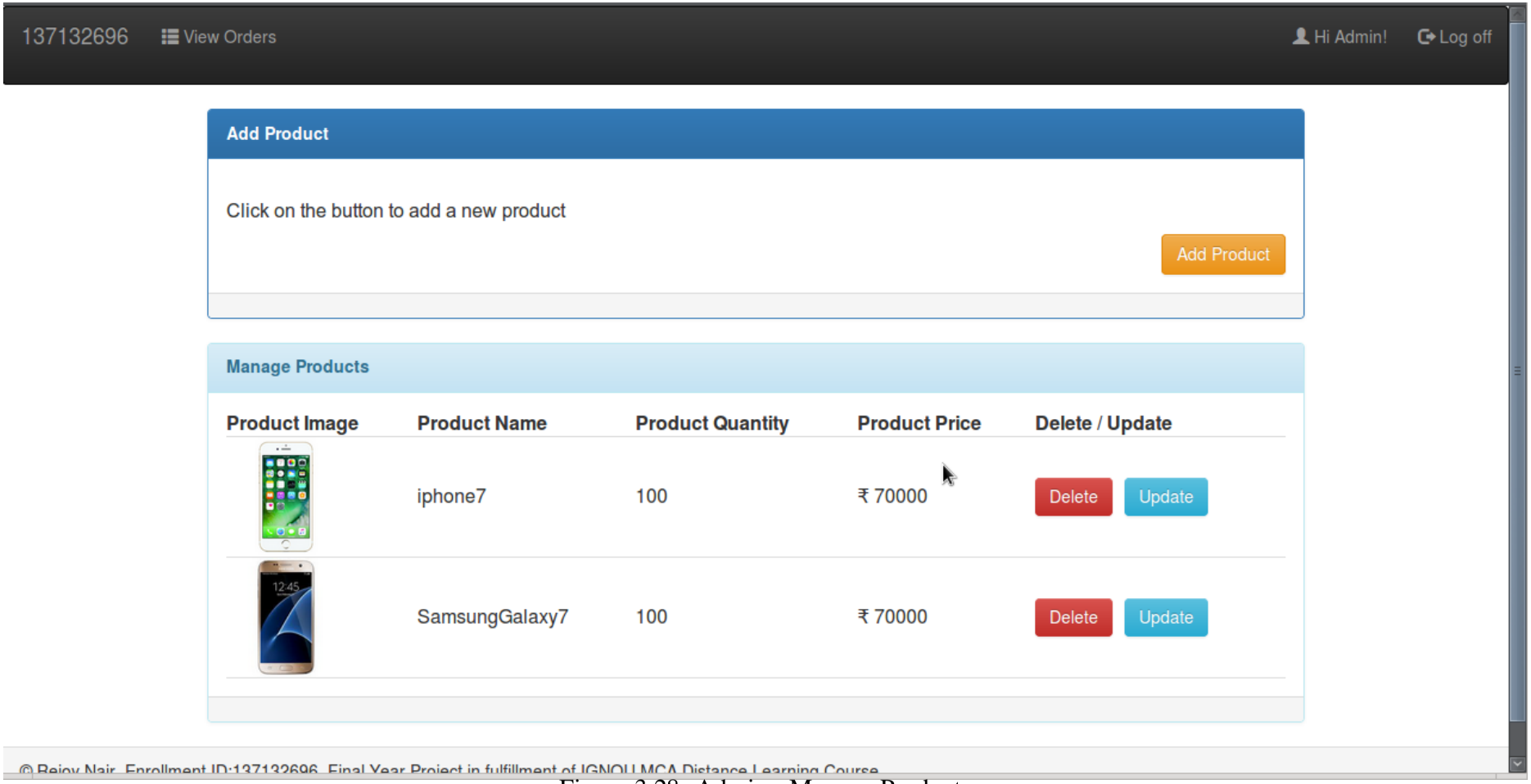


Figure 3.28: Admin - Manage Products

Table 3.10: **Admin Manage Products**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Banner	Banner to place the brand logo. Currently contains the enrollment ID 137132696
2.0	View Order	Link to navigate the Admin user back to the Admin Main View i.e., the View Order Data Page
3.0	User Profile icon	Displays the text 'Admin' indicating Admin user login
4.0	Log-off	Link to log-off the Admin User from the Admin User portal
5.0	Add Product	Button that navigates the admin user to the add product form
6.0	Manage Products - Product Image	Image of the Product that exists in the backend database
7.0	Manage Products - Product Name	Output text displaying the name of the Product that exists in the backend database
8.0	Manage Products - Product Quantity	Output text displaying the quantity of the Product as maintained in the backend database
9.0	Manage Products - Product Price	Output text displaying the price of the Product as maintained in the backend database

10.0	Manage Products – Delete Button	Button to delete the record of the corresponding product from the database
11.0	Manage Products – Update Button	Button to navigate the user to the update product form to update the record of the corresponding product from the database

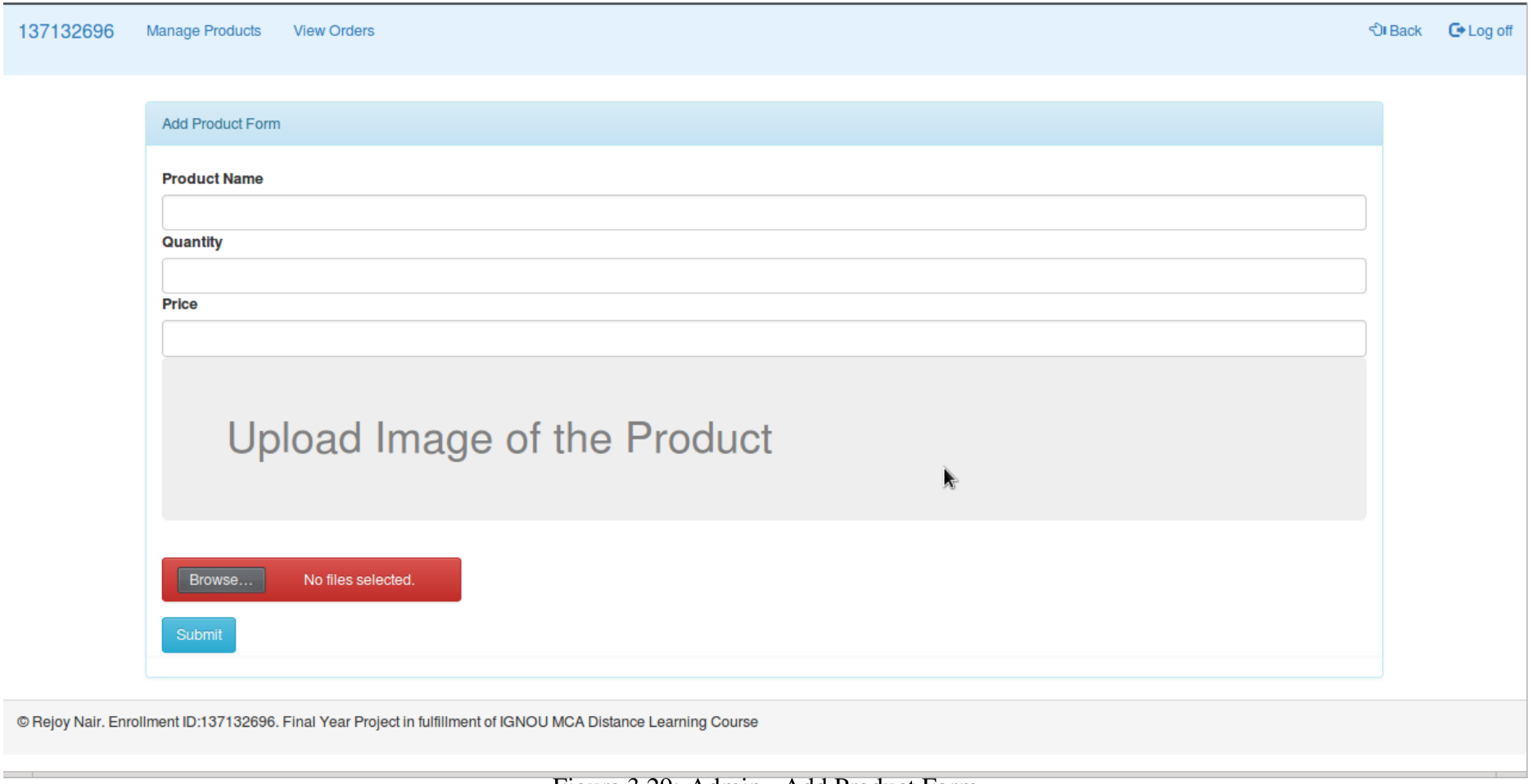


Figure 3.29: Admin - Add Product Form

Table 3.11: **Admin Add Products**

Sr. No.	**Input Element**	**Description & Behaviour**
1.0	Banner	Banner to place the brand logo. Currently contains the enrollment ID 137132696
2.0	View Order	Link to navigate the Admin user back to the Admin Main View i.e., the View Order Data Page
3.0	Manage Products	Link to navigate the Admin user back to the Admin – Manage Products Page
4.0	Back Button	Navigates the admin user back to the previous page
4.0	Log-off	Link to log-off the Admin User from the Admin User portal
5.0	Add Product – Product Name	Text field to input the name of the product to be added
6.0	Add Product – Quantity	Input field to input the name of the product to be added
7.0	Add Product – Price	Input field to input the price of the product to be added
8.0	Add Product – Upload Image File	File Upload utility to upload the image of the product to be added
9.0	Add Product – Submit Button	Button to post the Add Product form details to the server
10.0	Add Product – Alerts	Alerts to notify the admin users of fields that fail validation criteria

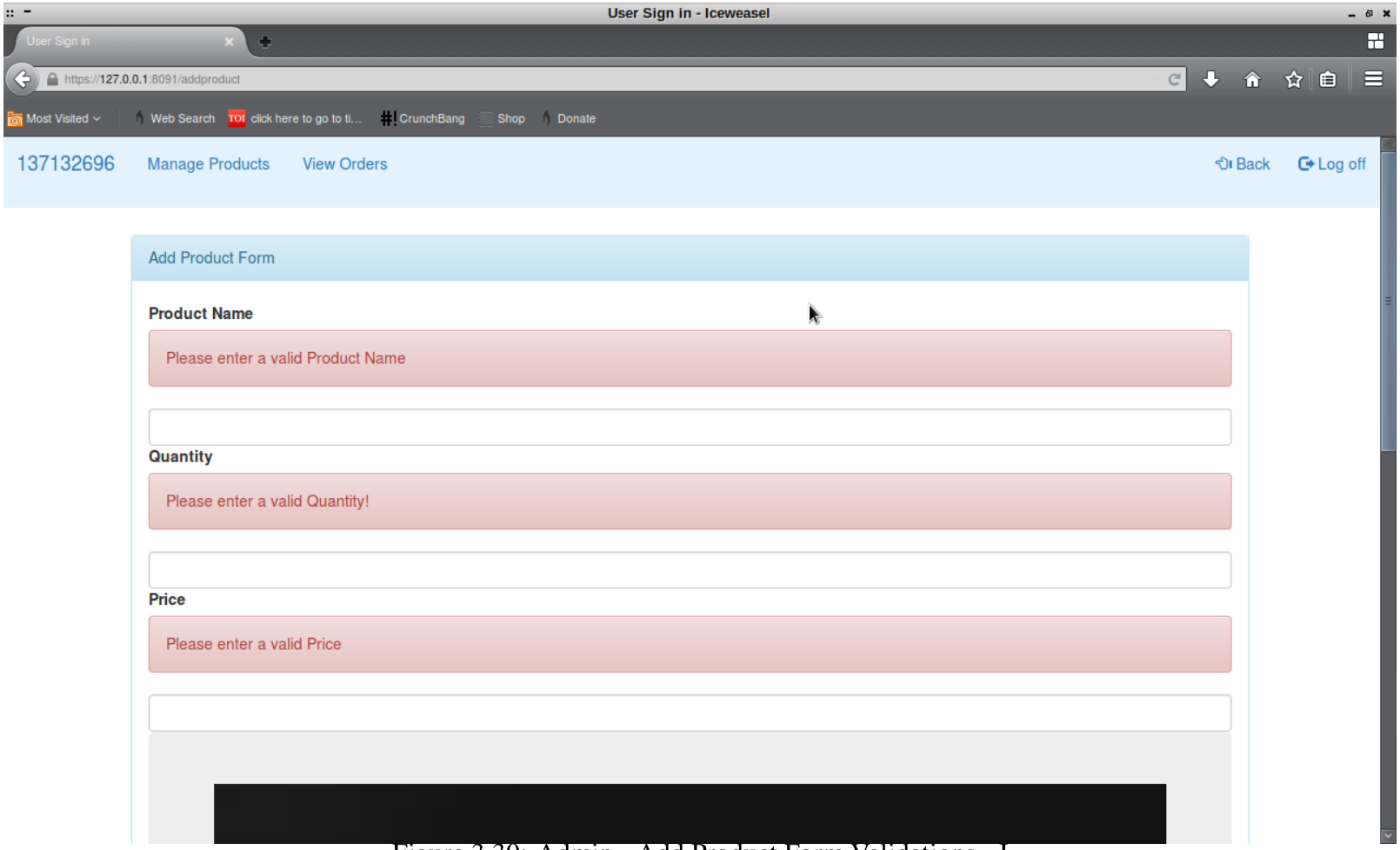


Figure 3.30: Admin - Add Product Form Validations - 1

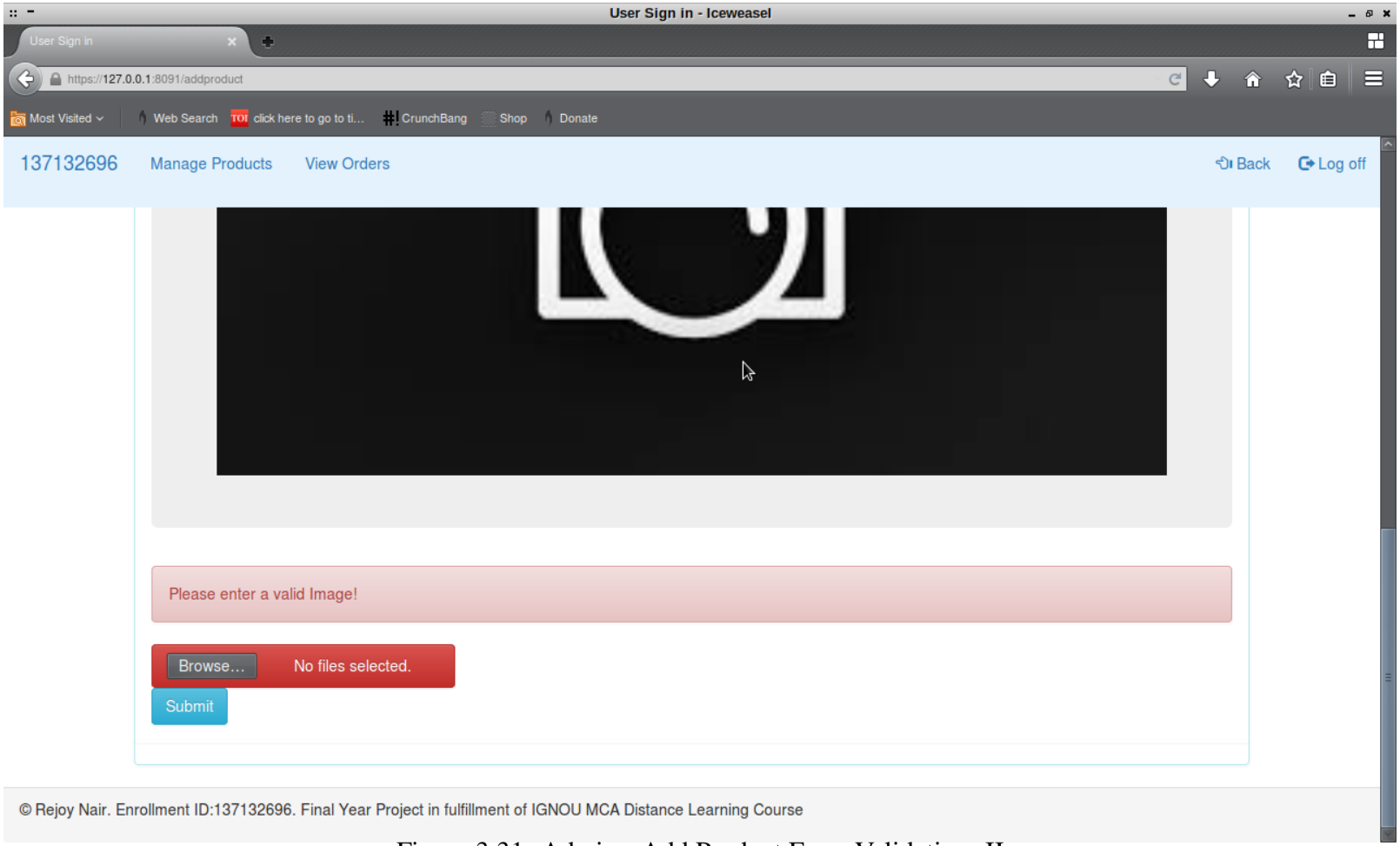


Figure 3.31: Admin - Add Product Form Validation -II

4 CODING

4.1 Go Code

```
// package for the main Go file
package main

// import packages
import (
    "encoding/json"
    "fmt"
    "html/template"
    "io"
    "net/http"
    "os"
    "strings"
    "strconv"
    "time"
    "log"

    "github.com/asaskevich/govalidator"
    "github.com/gorilla/mux"
    "github.com/gorilla/securecookie"
    "github.com/stripe/stripe-go"
    "github.com/stripe/stripe-go/charge"
)

// initialise variable to store cookie object
var cookieHandler = securecookie.New(
    securecookie.GenerateRandomKey(64),
    securecookie.GenerateRandomKey(32))

// initialise the router variable
var router = mux.NewRouter()


// function that initializes the entry point of the web app and renders the landing page
func IndexPage(w http.ResponseWriter, r *http.Request) {
    msg := GetMsg(w, r, "message")
    var u = &User{}
    u.Errors = make(map[string]string)
    if msg != "" {
        u.Errors["message"] = msg
        render(w, "signin", u)
    } else {
        u := &User{}
        render(w, "signin", u)
    }

}
```

```
47
48 // Login function to authenticate user and render ↩
     internal web page
49 func Login(w http.ResponseWriter, r *http.Request) {
50    name := r.FormValue("uname")
51    pass := r.FormValue("password")
52    u := &User{Username: name, Password: pass}
53    redirect := "/"
54    if name != "" && pass != "" {
55       if b, uuid := UserExists(u); b == true {
56          SetSession(&User{Uuid: uuid}, w)
57          if name != "admin" {
58             redirect = "/example"
59          } else {
60             redirect = "/admin"
61          }
62       } else {
63          SetMsg(w, "message", "please signup or enter a ↩
              valid username and password!")
64       }
65    } else {
66       SetMsg(w, "message", "Username or Password field ↩
          are empty!")
67    }
68    http.Redirect(w, r, redirect, 302)
69 }
70
71 // function Logout is to facilitate end of user's web ↩
     session and clear the user session cookies
72 func Logout(w http.ResponseWriter, r *http.Request) {
73    ClearSession(w, "session")
74    http.Redirect(w, r, "/", 302)
75 }
76
77 // func Examplepage is to retrieve data from the ↩
     database and send the data object to be rendered ↩
     after User login
78 func ExamplePage(w http.ResponseWriter, r *http.Request)↩
      {
79    uuid := GetUuid(r)
80    U := GetUserFromUuid(uuid)
81    if uuid != "" {
82       pdata, err := GetProduct()
83       if err != nil {
84          fmt.Println(err)
85       }
86
87       type Usr struct {
88          Id      string
89          Usrname string
90          Pwd     string
```

```
91          Fnm     string
92          Lnm     string
93          Eml     string
94      }
95
96      type Prdata struct {
97          Puid     string `json:"puid"`
98          Pname    string `json:"pname"`
99          Quantity string `json:"quantity"`
100         Price    string `json:"price"`
101         Image    string `json:"image"`
102     }
103
104     type Viewdata struct {
105         Usr Usr
106         Prd []Prdata
107     }
108
109     var Vd Viewdata
110
111     Vd.Usr.Id = U.Uuid
112     Vd.Usr.Usrname = U.Username
113     Vd.Usr.Pwd = U.Password
114     Vd.Usr.Fnm = U.Fname
115     Vd.Usr.Lnm = U.Lname
116     Vd.Usr.Eml = U.Email
117
118     // Pr := make([]Prdata, 0)
119     var Pr []Prdata
120
121     err = json.Unmarshal(pdata, &Pr)
122     if err != nil {
123         fmt.Println(err)
124     }
125
126     Vd.Prd = Pr
127
128     render(w, "internal", Vd)
129
130 } else {
131     SetMsg(w, "message", "Please login first!")
132     http.Redirect(w, r, "/", 302)
133 }
134 }
135
136 // function Signup is the accept user registration ↩
    details and save it to the database.
137 func Signup(w http.ResponseWriter, r *http.Request) {
138     switch r.Method {
139     case "GET":
140         u := &User{}
```

```
141         u.Errors = make(map[string]string)
142         u.Errors["lname"] = GetMsg(w, r, "lname")
143         u.Errors["fname"] = GetMsg(w, r, "fname")
144         u.Errors["email"] = GetMsg(w, r, "email")
145         u.Errors["username"] = GetMsg(w, r, "username")
146         u.Errors["password"] = GetMsg(w, r, "password")
147         render(w, "Signup", u)
148     case "POST":
149         if n := CheckUser(r.FormValue("userName")); n == ↩
                true {
150             SetMsg(w, "username", "User already exists. ↩
                   Please enter a unique username!")
151             http.Redirect(w, r, "/signup", 302)
152             return
153         }
154         u := &User{
155             Uuid:     Uuid(),
156             Fname:    r.FormValue("fName"),
157             Lname:    r.FormValue("lName"),
158             Email:    r.FormValue("email"),
159             Username: r.FormValue("userName"),
160             Password: r.FormValue("password"),
161         }
162         result, err := govalidator.ValidateStruct(u)
163         if err != nil {
164             e := err.Error()
165             if re := strings.Contains(e, "Lname"); re == ↩
                    true {
166                 SetMsg(w, "lname", "Please enter a valid ↩
                       Last Name")
167             }
168             if re := strings.Contains(e, "Email"); re == ↩
                    true {
169                 SetMsg(w, "email", "Please enter a valid ↩
                       Email Address!")
170             }
171             if re := strings.Contains(e, "Fname"); re == ↩
                    true {
172                 SetMsg(w, "fname", "Please enter a valid ↩
                       First Name")
173             }
174             if re := strings.Contains(e, "Username"); re ==↩
                     true {
175                 SetMsg(w, "username", "Please enter a valid ↩
                       Username!")
176             }
177             if re := strings.Contains(e, "Password"); re ==↩
                     true {
178                 SetMsg(w, "password", "Please enter a ↩
                       Password!")
179             }
```

```
180
181         }
182         if r.FormValue("password") != r.FormValue("↩
                cpassword") {
183             SetMsg(w, "password", "The passwords you ↩
                    entered do not Match!")
184             http.Redirect(w, r, "/signup", 302)
185             return
186         }
187
188         if result == true {
189             u.Password = EncryptPass(u.Password)
190             SaveData(u)
191             http.Redirect(w, r, "/", 302)
192             return
193         }
194         http.Redirect(w, r, "/signup", 302)
195
196     }
197 }
198
199 // function Admin renders the main view after admin user↩
       login.
200 // order details are retrieved from the database and ↩
      send as a data object to the client
201 func Admin(w http.ResponseWriter, r *http.Request) {
202     uuid := GetUuid(r)
203     if uuid != "" {
204
205         orw, err := GetOrders()
206         if err!=nil {
207             fmt.Println(err)
208         }
209
210         var or ([]map[string]interface{})
211
212         err = json.Unmarshal(orw, &or)
213         if err !=nil {
214             fmt.Println(err)
215         }
216
217         render(w, "Admin", or)
218     } else {
219         SetMsg(w, "message", "Please login first!")
220         http.Redirect(w, r, "/", 302)
221     }
222 }
223
224 // function ManageProduct provides the functionality to ↩
      view products and links for carrying out CRUD ↩
      operations
```

```
225 func Manageproduct(w http.ResponseWriter, r *http.↩
    Request) {
226     uuid := GetUuid(r)
227     // u := GetUserFromUuid(uuid)
228     if uuid != "" {
229         pdata, err := GetProduct()
230         if err != nil {
231             fmt.Println(err)
232         }
233 
234         var pr ([]map[string]interface{})
235 
236         err = json.Unmarshal(pdata, &pr)
237         if err != nil {
238             fmt.Println(err)
239         }
240 
241         render(w, "Manageproduct", pr)
242     } else {
243         SetMsg(w, "message", "Your session has expired. ↩
            Please login again!")
244         http.Redirect(w, r, "/", 302)
245     }
246 }
247 
248 // function Addproduct renders the add product form, ↩
    accepts admin users input and saves product data to ↩
    the database
249 func Addproduct(w http.ResponseWriter, r *http.Request) ↩
    {
250     switch r.Method {
251     case "GET":
252         uuid := GetUuid(r)
253         p := &Product{}
254         p.Errors = make(map[string]string)
255         p.Errors["pname"] = GetMsg(w, r, "pname")
256         p.Errors["quantity"] = GetMsg(w, r, "quantity")
257         p.Errors["price"] = GetMsg(w, r, "price")
258         p.Errors["image"] = GetMsg(w, r, "image")
259 
260         if uuid != "" {
261             render(w, "Addproduct", p)
262         } else {
263             SetMsg(w, "message", "Your Session has expired.↩
                Please login again!")
264             http.Redirect(w, r, "/", 302)
265         }
266 
267     case "POST":
268 
```

```
269         if n := CheckProduct(r.FormValue("productName")); n == true {
270             SetMsg(w, "pname", "Product already exists!")
271             http.Redirect(w, r, "/addproduct", 302)
272             return
273         }
274 
275         var pimage string
276 
277         r.Body = http.MaxBytesReader(w, r.Body, 2*1024*1024)
278         file, header, err := r.FormFile("productimage")
279         if err != nil {
280             // http.Error(w, err.Error(), http.StatusInternalServerError)
281             pimage = "NULL"
282         } else {
283             pimage = govalidator.ToString(header.Filename)
284         }
285 
286         p := &Product{
287             Puid:     Puid(),
288             Pname:    r.FormValue("productName"),
289             Quantity: r.FormValue("quantity"),
290             Price:    r.FormValue("price"),
291             Image:    pimage,
292         }
293 
294         result, err := govalidator.ValidateStruct(p)
295         if err != nil {
296             e := err.Error()
297             if re := strings.Contains(e, "Pname"); re == true {
298                 SetMsg(w, "pname", "Please enter a valid Product Name")
299             }
300             if re := strings.Contains(e, "Quantity"); re ==true {
301                 SetMsg(w, "quantity", "Please enter a valid Quantity!")
302             }
303             if re := strings.Contains(e, "Price"); re == true {
304                 SetMsg(w, "price", "Please enter a valid Price")
305             }
306             if re := strings.Contains(e, "Image") || pimage== "NULL"; re == true {
307                 SetMsg(w, "image", "Please enter a valid Image!")
308             }
```

```
309         }
310
311         if result == true {
312             f, err := os.Create("./files/" + p.Image)
313             if err != nil {
314                 // http.Error(w, err.Error(), http.↩
                        StatusInternalServerError)
315                 return
316             }
317             defer f.Close()
318             if _, err := io.Copy(f, file); err != nil {
319                 // http.Error(w, err.Error(), http.↩
                        StatusInternalServerError)
320                 return
321             }
322
323             SaveProductData(p)
324
325             http.Redirect(w, r, "/admin", 302)
326             return
327         }
328
329         http.Redirect(w, r, "/addproduct", 302)
330
331     }
332 }
333
334 //
335 func Showproduct(w http.ResponseWriter, r *http.Request)↩
        {
336     switch r.Method {
337     case "GET":
338         uuid := GetUuid(r)
339         fmt.Println(uuid)
340         fmt.Println(r.FormValue("prodid"))
341         pid := r.FormValue("prodid")
342         fmt.Println(pid)
343         a := GetProductFromPuid(pid)
344         a.Errors = make(map[string]string)
345         a.Errors["pname"] = GetMsg(w, r, "pname")
346         a.Errors["quantity"] = GetMsg(w, r, "quantity")
347         a.Errors["price"] = GetMsg(w, r, "price")
348         a.Errors["image"] = GetMsg(w, r, "image")
349         fmt.Println(a)
350         if uuid != "" {
351             render(w, "Showproduct", a)
352         } else {
353             SetMsg(w, "message", "Your Session has expired.↩
                    Please login again!")
354             http.Redirect(w, r, "/", 302)
355         }
```

```
356
357     case "POST":
358
359         fmt.Println("Post Request sent Sucessfully...")
360
361         if n := CheckProduct(r.FormValue("productName")); ↩
               n == true {
362             SetMsg(w, "pname", "Product already exists!")
363             http.Redirect(w, r, "/showproduct", 302)
364             return
365         }
366
367         var pimage string
368         pexists := 1
369
370         r.Body = http.MaxBytesReader(w, r.Body, ↩
               2*1024*1024)
371         file, header, err := r.FormFile("productimage")
372         if err != nil {
373             // http.Error(w, err.Error(), http.↩
                   StatusInternalServerError)
374             pimage =  r.FormValue("imgname")
375         } else {
376             pimage = govalidator.ToString(header.Filename)
377             pexists = 2
378         }
379
380         fmt.Println("Read file Sucessfully...")
381
382         pu := r.FormValue("prodid")
383
384         b := GetProductFromPuid(r.FormValue("prodid"))
385         fmt.Println(b.Pname)
386
387         b.Puid = pu
388
389         a := &Product{
390             Puid:     pu,
391             Pname:    r.FormValue("productName"),
392             Quantity: r.FormValue("quantity"),
393             Price:    r.FormValue("price"),
394             Image:    pimage,
395         }
396
397         fmt.Println(a)
398         a.Errors = make(map[string]string)
399
400         result, err := govalidator.ValidateStruct(a)
401         if err != nil {
402             e := err.Error()
```

```
403             if re := strings.Contains(e, "Pname"); re == ↩
                    true {
404                 // SetMsg(w, "pname", "Please enter a valid ↩
                        Product Name")
405                 a.Errors["pname"] = "Please enter a valid ↩
                        Product Name"
406                 a.Pname = b.Pname
407             }
408             if re := strings.Contains(e, "Quantity"); re ==↩
                     true {
409                 // SetMsg(w, "quantity", "Please enter a ↩
                        valid Quantity!")
410                 a.Errors["quantity"] = "Please enter a valid↩
                         Quantity!"
411                 a.Quantity = b.Quantity
412             }
413             if re := strings.Contains(e, "Price"); re == ↩
                    true {
414                 // SetMsg(w, "price", "Please enter a valid ↩
                        Price")
415                 a.Errors["price"] = "Please enter a valid ↩
                        Price"
416                 a.Price = b.Price
417             }
418             if re := strings.Contains(e, "Image") || pimage↩
                     == "NULL"; re == true {
419                 // SetMsg(w, "image", "Please enter a valid ↩
                        Image!")
420                 a.Errors["image"] = "Please enter a valid ↩
                        Image!"
421             }
422         }
423 
424         fmt.Println(result)
425 
426         if result == true {
427             if pexists == 2 {
428                 fmt.Println(pexists)
429                 f, err := os.Create("./files/" + a.Image)
430                 if err != nil {
431                     // http.Error(w, err.Error(), http.↩
                            StatusInternalServerError)
432                     return
433                 }
434                 defer f.Close()
435                 if _, err := io.Copy(f, file); err != nil {
436                     // http.Error(w, err.Error(), http.↩
                            StatusInternalServerError)
437                     return
438                 }
439 
```

```
440             fmt.Println("Saved File Image...")
441          }
442
443          UpdateProductData(a)
444
445          http.Redirect(w, r, "/manageproduct", 302)
446          return
447       }
448
449       // http.Redirect(w, r, "/showproduct", 302)
450
451       uuid := GetUuid(r)
452       if uuid != "" {
453          render(w, "Showproduct", a)
454       } else {
455          SetMsg(w, "message", "Your Session has expired.↩
                Please login again!")
456          http.Redirect(w, r, "/", 302)
457       }
458
459    }
460 }
461
462
463 func Deleteproduct(w http.ResponseWriter, r *http.↩
    Request) {
464    uuid := GetUuid(r)
465    if uuid != "" {
466       pid := r.FormValue("proid")
467       fmt.Println(pid)
468       DeleteProductData(pid)
469       fmt.Println("Deleting Product Entries ...")
470       http.Redirect(w, r, "/manageproduct", 302)
471    }
472 }
473
474 // function Vieworder retrieves from the database the ↩
    single order data object and renders it on the admin ↩
    page
475 func Vieworder(w http.ResponseWriter, r *http.Request) {
476    uuid := GetUuid(r)
477    // u := GetUserFromUuid(uuid)
478    if uuid != "" {
479       oid := r.FormValue("orderid")
480       fmt.Println(oid)
481       or, err := GetOrderFromOuid(oid)
482       if err !=nil {
483          fmt.Println(err)
484       }
485
486       render(w, "Vieworder", or)
```

```
487
488     } else {
489         SetMsg(w, "message", "Your session has expired. ↩
                Please login again!")
490         http.Redirect(w, r, "/", 302)
491     }
492 }
493
494 // function Placeorder renders the place order page that↩
       displays the order basket and provides the link to ↩
      make payment
495 func PlaceOrder(w http.ResponseWriter, r *http.Request) ↩
      {
496     uuid := GetUuid(r)
497     pubkey := "pk_test_75gzALA1yuRnl1y7qWdimlov"
498     if uuid != "" {
499         render(w, "Placeorder", pubkey)
500     } else {
501         SetMsg(w, "message", "Your session has expired. ↩
                Please login again!")
502         http.Redirect(w, r, "/", 302)
503     }
504 }
505
506 // function Payment provides the functionality to call ↩
      Stripe's API for payment, collect shipping details, ↩
      post charge details,
507 // generate the order and save order details to the ↩
      database
508 // order details contain the product details and ↩
      shipping details
509 func Payment(w http.ResponseWriter, r *http.Request) {
510     uuid := GetUuid(r)
511     if uuid != "" {
512         emaddress := GetUserEmailFromUuid(uuid)
513
514         stripe.Key = "sk_test_rmp1OSLAWeSj1cAbJ7CvG3Rl"
515         token := r.FormValue("stripeToken")
516
517         var m = make(map[string]interface{})
518
519         m["eladdress"] = r.FormValue("stripeEmail")
520         m["shaddressname"] = r.FormValue("↩
                stripeShippingName")
521         m["shaddressline"] = r.FormValue("↩
                stripeShippingAddressLine1")
522         m["shaddresszip"] = r.FormValue("↩
                stripeShippingAddressZip")
523         // m["shaddressstate"] = r.FormValue("↩
                stripeShippingAddressState")
```

```
524         m["shaddresscity"] = r.FormValue("↩
                stripeShippingAddressCity")
525         m["shaddresscountry"] = r.FormValue("↩
                stripeShippingAddressCountry")
526
527
528         // marshal data to json
529         j, err := json.Marshal(m)
530         if err != nil {
531            fmt.Println(err)
532         }
533
534         s := r.FormValue("totalprice")
535         totalamount, err := strconv.ParseUint(s, 10, 64)
536            if err != nil {
537               panic(err)
538         }
539
540         params := &stripe.ChargeParams{
541            Amount: totalamount,
542            Currency: "inr",
543            Desc: "Order Payment",
544         }
545         params.SetSource(token)
546
547         charge, err := charge.New(params)
548         if err != nil {
549            fmt.Println(err)
550         }
551
552         tmst := strconv.FormatInt(time.Now().Unix(), 10)
553
554         fmt.Println(tmst)
555
556         O := &Order{
557            Ouid:              Ouid(),
558            Userid:            uuid,
559            Token:             r.FormValue("stripeToken"),
560            OrderDetail:       r.FormValue("Output"),
561            ShipDetail:        string(j),
562            TotalAmount:         r.FormValue("totalprice"),
563            ChargedAmount :     charge.Amount,
564            Chargeid:          charge.ID,
565            ChargeStatus:      charge.Status,
566            Timestamp:         tmst,
567         }
568
569         if charge.Status == "succeeded" {
570            err := SaveOrderData(O)
571            if err != nil {
572               fmt.Println(err)
```

```
573             }
574
575             OrderEmail(r.FormValue("stripeEmail"))
576             if emaddress != r.FormValue("stripeEmail") {
577                OrderEmail(emaddress)
578             }
579
580             or, err := GetOrderFromOuid(O.Ouid)
581             if err !=nil {
582                   fmt.Println(err)
583             }
584
585             render(w, "Payment", or)
586
587          } else {
588             render(w, "Payment", "Transaction Failed. ↩
                  Please Try Again!")
589          }
590
591       } else {
592          SetMsg(w, "message", "Your session has expired. ↩
             Please login again!")
593          http.Redirect(w, r, "/", 302)
594       }
595 }
596
597 func render(w http.ResponseWriter, name string, data ↩
       interface{}) {
598       tmpl, err := template.ParseGlob("view/*.html")
599       if err != nil {
600          http.Error(w, err.Error(), http.↩
             StatusInternalServerError)
601       }
602       tmpl.ExecuteTemplate(w, name, data)
603 }
604
605 func main() {
606       govalidator.SetFieldsRequiredByDefault(true)
607       http.Handle("/initializr/", http.StripPrefix("/↩
          initializr/", http.FileServer(http.Dir("initializr↩
          "))))
608       http.Handle("/files/", http.StripPrefix("/files/", ↩
          http.FileServer(http.Dir("files"))))
609       router.HandleFunc("/", IndexPage)
610       router.HandleFunc("/login", Login).Methods("POST")
611       router.HandleFunc("/logout", Logout).Methods("POST")
612       router.HandleFunc("/example", ExamplePage)
613       router.HandleFunc("/signup", Signup).Methods("POST", ↩
          "GET")
614       router.HandleFunc("/admin", Admin)
615       router.HandleFunc("/manageproduct", Manageproduct)
```

```
616     router.HandleFunc("/addproduct", Addproduct).Methods(↩
            "POST", "GET")
617     router.HandleFunc("/showproduct", Showproduct)
618     router.HandleFunc("/deleteproduct", Deleteproduct)
619     router.HandleFunc("/vieworder", Vieworder)
620     router.HandleFunc("/placeorder", PlaceOrder)
621     router.HandleFunc("/payment", Payment)
622     http.Handle("/", router)
623     // http.ListenAndServe(":8090", nil)
624     go http.ListenAndServe(":8090", http.RedirectHandler(↩
            "https://127.0.0.1:8091",301))
625     err := http.ListenAndServeTLS(":8091", "tls/cert.pem"↩
            , "tls/key.pem", nil)
626     log.Fatal(err)
627 }
```

```go
//package User for managing User data
package main

// import packages
import (
    "database/sql"
    "fmt"

    _ "github.com/mattn/go-sqlite3"
    "github.com/satori/go.uuid"
    "golang.org/x/crypto/bcrypt"
)

// User Model
type User struct {
    Uuid     string            `valid:"required,uuidv4"`
    Username string            `valid:"required,alphanum"`
    Password string            `valid:"required"`
    Fname    string            `valid:"required,alpha"`
    Lname    string            `valid:"required,alpha"`
    Email    string            `valid:"required,email"`
    Errors   map[string]string `valid:"-"`
}

// function SaveData saves the accepted user enetered data to the database
func SaveData(u *User) error {
    var db, _ = sql.Open("sqlite3", "cache/users.sqlite3")
    defer db.Close()
    db.Exec("create table if not exists users (uuid text not null unique, firstname text not null, lastname text not null, username text not null unique, email text not null unique, password text not null, primary key(uuid))")
    tx, _ := db.Begin()
    stmt, _ := tx.Prepare("insert into users (uuid, firstname, lastname, username, email, password) values (?, ?, ?, ?, ?, ?)")
    _, err := stmt.Exec(u.Uuid, u.Fname, u.Lname, u.Username, u.Email, u.Password)
    tx.Commit()
    return err
}

// function UserExists is used to check if a user provided username exists in the database and its subsequent authentication
// If the username exists in the database, the user
```

```
    provided password is compared with the password saved in the database
func UserExists(u *User) (bool, string) {
    var db, _ = sql.Open("sqlite3", "cache/users.sqlite3")
    defer db.Close()
    var ps, uu string
    q, err := db.Query("select uuid, password from users where username = '" + u.Username + "'")
    if err != nil {
        return false, ""
    }
    for q.Next() {
        q.Scan(&uu, &ps)
    }
    pw := bcrypt.CompareHashAndPassword([]byte(ps), []byte(u.Password))
    if uu != "" && pw == nil {
        return true, uu
    }
    return false, ""
}

// function CheckUser is used to check if a user provided username exists in the database
func CheckUser(user string) bool {
    if user =="" {
        return false
    }
    var db, _ = sql.Open("sqlite3", "cache/users.sqlite3")
    defer db.Close()
    var un string
    q, err := db.Query("select username from users where username = '" + user + "'")
    if err != nil {
        return false
    }
    {
    for q.Next() {
        q.Scan(&un)
        }
    }
    if un == user {
        return true
    }
    return false
}

// function GetUserFromUuid is used to retrieve from the database user information corresponding to the UUID
```

```
81 func GetUserFromUuid(uuid string) *User {
82    var db, _ = sql.Open("sqlite3", "cache/users.sqlite3"↩
         )
83    defer db.Close()
84    var uu, fn, ln, un, em, pass string
85    q, err := db.Query("select * from users where uuid = ↩
         '" + uuid + "'")
86    if err != nil {
87       return &User{}
88    }
89    for q.Next() {
90       q.Scan(&uu, &fn, &ln, &un, &em, &pass)
91    }
92    return &User{Username: un, Fname: fn, Lname: ln, ↩
         Email: em, Password: pass}
93 }
94
95 // function getUserEmailFromUuid is the used to retrieve↩
      from the database the email address corresponding to↩
      the user's UUID
96 func GetUserEmailFromUuid(uuid string) string {
97    var db, _ = sql.Open("sqlite3", "cache/users.sqlite3"↩
         )
98    defer db.Close()
99    var em string
100   q, err := db.Query("select email from users where ↩
         uuid = '" + uuid + "'")
101   if err != nil {
102      return em
103   }
104   for q.Next() {
105      q.Scan(&em)
106   }
107   return em
108 }
109
110 // function EncryptPass is used to encrypt user password
111 func EncryptPass(password string) string {
112   pass := []byte(password)
113   hashpw, _ := bcrypt.GenerateFromPassword(pass, bcrypt↩
         .DefaultCost)
114   return string(hashpw)
115 }
116
117 // function Uuid is used to generate the UUID string for↩
      a new user
118 func Uuid() string {
119   id := uuid.NewV4()
120   return id.String()
121 }
```

```go
// package Webcookie for managing sessions
package main

// import package
import "net/http"

// function SetSession creates a cookie for a user web session after successful user login
func SetSession(u *User, w http.ResponseWriter) {
    value := map[string]string{
        "uuid": u.Uuid,
    }
    if encoded, err := cookieHandler.Encode("session", value); err == nil {
        cookie := &http.Cookie{
            Name:  "session",
            Value: encoded,
            Path:  "/",
        }
        http.SetCookie(w, cookie)
    }
}

// function GetUuid retrieves the uuid string from the cookie by decoding the "value" parameter of the cookie
func GetUuid(r *http.Request) (uuid string) {
    if cookie, err := r.Cookie("session"); err == nil {
        cookieValue := make(map[string]string)
        if err = cookieHandler.Decode("session", cookie.Value, &cookieValue); err == nil {
            uuid = cookieValue["uuid"]
        }
    }
    return uuid
}

// function ClearSession clears the cookie created for the user web session
func ClearSession(w http.ResponseWriter, name string) {
    cookie := &http.Cookie{
        Name:   name,
        Value:  "",
        Path:   "/",
        MaxAge: -1,
    }
    http.SetCookie(w, cookie)
}

// function GetMsg is used decode the Alert message fron
```

```
     the Cookie "value" parameter and send it to the ↩
    client application
45 func GetMsg(w http.ResponseWriter, r *http.Request, name↩
     string) (msg string) {
46    if cookie, err := r.Cookie(name); err == nil {
47       cookieValue := make(map[string]string)
48       if err = cookieHandler.Decode(name, cookie.Value, ↩
            &cookieValue); err == nil {
49          msg = cookieValue[name]
50          ClearSession(w, name)
51       }
52    }
53
54    return msg
55 }
56
57 // function SetMsg is used to receive a message and ↩
    create the alerts for the particular user web session↩
    .
58 // This is done by encoding the message into the cookie ↩
    value parameter.
59 func SetMsg(w http.ResponseWriter, name string, msg ↩
    string) {
60    value := map[string]string{
61       name: msg,
62    }
63    if encoded, err := cookieHandler.Encode(name, value);↩
         err == nil {
64       cookie := &http.Cookie{
65          Name:  name,
66          Value: encoded,
67          Path:  "/",
68       }
69       http.SetCookie(w, cookie)
70    }
71 }
```

```go
// package Product for managing product data
package main

// import packages
import (
    "database/sql"
    "encoding/json"
    _ "github.com/mattn/go-sqlite3"
    uuid "github.com/satori/go.uuid"
)

// Product Model
type Product struct {
    Puid     string            `valid:"required,uuidv4"`
    Pname    string            `valid:"required,alphanum"`
    Quantity string            `valid:"required,numeric"`
    Price    string            `valid:"required,numeric"`
    Image    string            `valid:"-"`
    Errors   map[string]string `valid:"-"`
}


// function SaveProductData saves admin user entered Product data that is validated  to the database
func SaveProductData(p *Product) error {
    var db, _ = sql.Open("sqlite3", "cache/users.sqlite3")
    defer db.Close()
    db.Exec("create table if not exists products (puid text not null unique, pname text not null, quantity text not null, price text not null , image text, primary key(puid))")
    tx, _ := db.Begin()
    stmt, _ := tx.Prepare("insert into products (puid, pname, quantity, price, image ) values (?, ?, ?, ?, ? )")
    _, err := stmt.Exec(p.Puid, p.Pname, p.Quantity, p.Price, p.Image)
    tx.Commit()
    return err
}

// function ProductExists is used to retrieve the Puid and Name parameter values of product information for the specific product from the database
func ProductExists(p *Product) (bool, string) {
    var db, _ = sql.Open("sqlite3", "cache/users.sqlite3")
    defer db.Close()
```

```
39      var pid, pn string
40      q, err := db.Query("select puid, pname from products ↩
           where pname = '" + p.Pname + "'")
41      if err != nil {
42          return false, ""
43      }
44      for q.Next() {
45          q.Scan(&pid, &pn)
46      }
47      if pid != "" && pn != "" {
48          return true, pid
49      }
50      return false, ""
51  }
52
53  // function CheckProduct is used to check if product ↩
        information for the specific product exists in the ↩
        database using the name parameter
54  func CheckProduct(product string) bool {
55      var db, _ = sql.Open("sqlite3", "cache/users.sqlite3"↩
           )
56      defer db.Close()
57      var pr string
58      q, err := db.Query("select product from products ↩
           where pname = '" + product + "'")
59      if err != nil {
60          return false
61      }
62      for q.Next() {
63          q.Scan(&pr)
64      }
65      if pr == product {
66          return true
67      }
68      return false
69  }
70
71  //function GetProductFromPuid is used to retrieve from ↩
        the database product information corresponding to the↩
         Product UUID
72  func GetProductFromPuid(puid string) *Product {
73      var db, _ = sql.Open("sqlite3", "cache/users.sqlite3"↩
           )
74      defer db.Close()
75      var pid, product, quantity, price, image string
76      q, err := db.Query("select * from products where puid↩
            = '" + puid + "'")
77      if err != nil {
78          return &Product{}
79      }
80      for q.Next() {
```

```
81          q.Scan(&pid, &product, &quantity, &price, &image)
82      }
83      return &Product{Pname: product, Quantity: quantity, ↩
            Price: price, Image: image}
84  }
85
86  // function Puid is used to generate the UUID string for↩
        a new product
87  func Puid() string {
88      id := uuid.NewV4()
89      return id.String()
90  }
91
92
93  // function GetProduct is used to retrieve from the ↩
        database the entire product table and encode it to ↩
        JSON
94  func GetProduct() ([]byte, error) {
95      var db, _ = sql.Open("sqlite3", "cache/users.sqlite3"↩
            )
96      defer db.Close()
97      db.Exec("create table if not exists delprod (puid ↩
            text not null unique, primary key(puid))")
98      // var pid, product, quantity, price, image string
99      rows, err := db.Query("select * from products where ↩
            not exists (select delprod.puid from delprod where↩
             products.puid = delprod.puid)")
100     if err != nil {
101         return nil, err
102     }
103     defer rows.Close()
104     columns, err := rows.Columns()
105     if err != nil {
106         return nil, err
107     }
108     count := len(columns)
109     tableData := make([]map[string]interface{}, 0)
110     values := make([]interface{}, count)
111     valuePtrs := make([]interface{}, count)
112     for rows.Next() {
113         for i := 0; i < count; i++ {
114             valuePtrs[i] = &values[i]
115         }
116         rows.Scan(valuePtrs...)
117         entry := make(map[string]interface{})
118         for i, col := range columns {
119             var v interface{}
120             val := values[i]
121             b, ok := val.([]byte)
122             if ok {
123                 v = string(b)
```

```
124             } else {
125                 v = val
126             }
127             entry[col] = v
128         }
129         tableData = append(tableData, entry)
130     }
131     jsonData, err := json.Marshal(tableData)
132     if err != nil {
133         return nil, err
134     }
135         return jsonData, nil
136 }
137
138 // function DeleteProductData saves admin user deleted ↩
    Product data that is validated  to the database
139 func DeleteProductData(pid string) error {
140     var db, _ = sql.Open("sqlite3", "cache/users.sqlite3"↩
        )
141     defer db.Close()
142     // db.Exec("create table if not exists delprod (puid ↩
        text not null unique, primary key(puid))")
143     tx, _ := db.Begin()
144     stmt, _ := tx.Prepare("insert into delprod (puid) ↩
        values (?)")
145     _, err := stmt.Exec(pid)
146     tx.Commit()
147     return err
148 }
149
150
151 // function UpdateProductData saves admin user entered ↩
    Product data that is validated  to the database
152 func UpdateProductData(p *Product) error {
153     var db, _ = sql.Open("sqlite3", "cache/users.sqlite3"↩
        )
154     defer db.Close()
155     tx, _ := db.Begin()
156     // stmt, _ := tx.Prepare("REPLACE INTO products (↩
        pname, quantity, price, image ) values (?, ?, ?, ?↩
         ) where products.puid = '" + p.Puid + "'")
157     stmt, _ := tx.Prepare("REPLACE INTO products (puid, ↩
        pname, quantity, price, image ) values (?, ?, ?, ↩
        ?, ? )")
158     _, err := stmt.Exec(p.Puid, p.Pname, p.Quantity, p.↩
        Price, p.Image)
159     tx.Commit()
160     return err
161 }
```

```go
// package Order to manage Order data
package main

// import packages
import (
    "database/sql"
    "encoding/json"
    "time"
    "fmt"
    _ "github.com/mattn/go-sqlite3"
    uuid "github.com/satori/go.uuid"
)

// OrderShipAdd Model for Order Shipping data
type OrderShipAdd struct {
        eladdress            string
        shaddressname        string
        shaddressline        string
        shaddresszip         string
        shaddresscity        string
        shaddresscountry     string
}

// Order Model
type Order struct {
    Ouid           string                    `valid:"required,uuidv4"`
    Userid         string                  `valid:"required,alphanum"`
    Token          string                  `valid:"required,alphanum"`
    OrderDetail     string                 `valid:"-"`
    ShipDetail     string                  `valid:"-"`
    TotalAmount     string                 `valid:"-"`
    ChargedAmount   uint64                 `valid:"-"`
    Chargeid     string                  `valid:"-"`
    ChargeStatus   string                  `valid:"-"`
    Timestamp      string                  `valid:"-"`
    Errors         map[string]string          `valid:"-"`
}

// function SaveOrder is used to save the Order data to the database
func SaveOrderData(o *Order) error {
    var db, _ = sql.Open("sqlite3", "cache/users.sqlite3")
    defer db.Close()
    db.Exec("create table if not exists orders (ouid text not null unique, userid text not null, token text not null, orderdetail blob not null, shipdetail
```

```
        blob not null, totalamount text not null, ↩
        chargedamount integer not null, chargeid text not ↩
        null, chargestatus text not null, timestamp text, ↩
        primary key(ouid))")
44    tx, _ := db.Begin()
45    stmt, _ := tx.Prepare("insert into orders (ouid, ↩
        userid, token, orderdetail, shipdetail, ↩
        totalamount, chargedamount, chargeid, chargestatus↩
        , timestamp) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)↩
        ")
46    _, err := stmt.Exec(o.Ouid, o.Userid, o.Token, o.↩
        OrderDetail, o.ShipDetail, o.TotalAmount, o.↩
        ChargedAmount, o.Chargeid, o.ChargeStatus, o.↩
        Timestamp)
47    tx.Commit()
48    return err
49 }
50
51 // function Ouid is used to generate the OUID string for↩
       a new Order
52 func Ouid() string {
53    id := uuid.NewV4()
54    return id.String()
55 }
56
57
58 // function GetOrders is used to retrieve from the ↩
     Orders table of the database the Order UUID, Order ↩
     Value and Order Timestamp
59 // for all orders in the entire product table and encode↩
       it to JSON
60 func GetOrders() ([]byte, error) {
61    var db, _ = sql.Open("sqlite3", "cache/users.sqlite3"↩
        )
62    defer db.Close()
63    var ou string
64    var ta, ts int64
65    q, err := db.Query("select ouid, chargedamount, ↩
        timestamp from orders")
66    if err != nil {
67       fmt.Println(err)
68    }
69
70    var a []interface{}
71
72    for q.Next() {
73       q.Scan(&ou, &ta, &ts)
74       b := make(map[string]interface{})
75       b["ouid"] = ou
76       b["chargedamount"] = float64(ta)/100
77       // b["timestamp"] = ts
```

```
78          b["timestamp"] = string(time.Unix(ts, 0).Format("↩
               02.01.2006 15:04:05"))
79          a = append(a, b)
80      }
81
82      getord, err := json.Marshal(a)
83      if err != nil {
84          return nil, err
85      }
86          return getord, nil
87  }
88
89  // function GetOrderFromOuid is used to retrieve from ↩
        the database the entire order data for the Order UUID↩
         table and encode it to JSON
90  func GetOrderFromOuid(ouid string) (map[string]interface↩
        {}, error) {
91      var db, _ = sql.Open("sqlite3", "cache/users.sqlite3"↩
            )
92      defer db.Close()
93
94      var ou, od, sh string
95      var ta, ts int64
96      // var od []byte
97
98      q, err := db.Query("select ouid, orderdetail, ↩
            shipdetail, chargedamount, timestamp from orders ↩
            where ouid = '" + ouid + "'")
99      if err != nil {
100         return nil, err
101     }
102
103     getord := make(map[string]interface{})
104
105     for q.Next() {
106         q.Scan(&ou, &od, &sh, &ta, &ts)
107         getord["ouid"] = ou
108         getord["orderdetail"] = od
109         getord["shipdetail"] = sh
110         getord["chargedamount"] = float64(ta/100)
111         getord["timestamp"] = string(time.Unix(ts, 0).↩
               Format("02.01.2006 15:04:05"))
112     }
113
114     var odd ([]map[string]interface{})
115
116     err = json.Unmarshal([]byte(od), &odd)
117     if err != nil {
118         fmt.Println(err)
119     }
120
```

```
121     getord["orderdetail"] = odd
122
123
124     osd := make(map[string]interface{})
125
126     err = json.Unmarshal([]byte(sh), &osd)
127     if err != nil {
128         fmt.Println(err)
129     }
130
131     getord["shipdetail"] = osd
132
133     return getord, nil
134
135 }
```

```
// package EmailNot for email notification
package main

// import packages
import (
   "log"
   "net/smtp"
)

// function OrderEmail triggers an email notification after an order is sucessfully placed
func OrderEmail(el string) {
        from := "123redoc@gmail.com"
    password := "xxxxxxxxx"

   auth := smtp.PlainAuth("", from, password, "smtp.gmail.com")
   to := el
   msg := "From: " + from + "\r\n" +
            "To: " + to + "\r\n" +
            "Subject: Order Confirmation" + "\r\n\r\n" +
            "You have sucessfully placed your order with Mobile Store" + "\r\n" +
          "Sincerely" + "\r\n" +
          "The Online Mobile Store" + "\r\n" +
          "IGNOU Enrollment ID: 137132696" + "\r\n" + "\r\n\r\n" +
          "This is a test email of a academic project" + "\r\n"
    /* Ports 465 and 587 are intended for email client to email server communication (sending email). Port 465 is for smtps.
    SSL encryption is started automatically before any SMTP level communication. Port 587 is for msa. It is almost like standard SMTP port. */

   err := smtp.SendMail("smtp.gmail.com:587", auth, from, []string{to}, []byte(msg))
   if err != nil {
      log.Printf("Error: %s", err)
        return
   }
   log.Print("message sent")
}
```

4.2 HTML Code

```
{{ define "header"}}
<html>
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge,
   chrome=1">
<title>User Sign in</title>
<meta name="description" content="">
<meta name="viewport" content="width=device-width,
   initial-scale=1">

<link rel="stylesheet" href="/initializr/css/
   bootstrap.min.css">
<link rel="stylesheet" href="/initializr/css/
   bootstrap-theme.min.css">
<link rel="stylesheet" href="/initializr/css/main.css">

<script src="/initializr/js/vendor/
   modernizr-2.8.3-respond-1.4.2.min.js"></script>

<script src="//ajax.googleapis.com/ajax/libs/jquery
   /1.11.3/jquery.min.js"></script>
<script>window.jQuery || document.write('<script src="js/
   vendor/jquery-1.11.3.min.js"><\/script>')</script>

<script src="/initializr/js/vendor/bootstrap.min.js">
   </script>

<script src="/initializr/js/main.js"></script>

</head>
<body>
{{end}}



{{define "footer"}}
<footer class="panel-footer" ><p>&copy; Rejoy Nair.
   Enrollment ID:137132696. Final Year Project in
   fulfillment of IGNOU MCA Distance Learning Course</p><
   /footer>
<!-- <footer class="panel-footer" style="
   position:absolute;bottom:0;width:100%;" ><p>&copy;
   Rejoy Nair. Enrollment ID:137132696. Final Year
   Project in fulfillment of IGNOU MCA Distance Learning
   Course</p></footer> -->


<!-- Google Analytics: change UA-XXXXX-X to be your site'
```

```
    s ID. -->
<script>
            (function(b,o,i,l,e,r){
               b.GoogleAnalyticsObject=l;b[l]||(b[l]=
            function(){(b[l].q=b[l].q||[]).push(arguments
               )});b[l].l=+new Date;
            e=o.createElement(i);r=o.getElementsByTagName
               (i)[0];
            e.src='//www.google-analytics.com/
               analytics.js';
            r.parentNode.insertBefore(e,r)}(window,
               document,'script','ga'));
            ga('create','UA-XXXXX-X','auto');ga('send','
               pageview');
</script>
</body>
</html>
{{end}}
```

```
{{define "signin"}}
{{template "header" .}}
<h1 class="alert alert-info">Login</h1>
<div class="container">
    {{with .Errors.message}}
    <div class="alert alert-danger">
        {{.}}
    </div>
    {{end}}
    <form method="POST" action="/login">
        <label class="form-control" for="uname">User
            Name</label>
        <input class="form-control" type="text" id="uname
            " name="uname">
        <label class="form-control" for="password">
            Password</label>
        <input class="form-control" type="password" id="
            password" name="password">
        <button class="btn btn-info" type="submit">
            Submit</button>
    </form>
</div>
<div class="alert alert-info" role="alert">
    <a class="alert-link" href="/signup">Sign Up</a>
</div>
{{template "footer" .}}
{{end}}
```

```
{{define "Signup"}}
{{template "header" .}}
<h1 class="alert alert-info">Sign Up</h1>
<div class="container">
    <form  method="POST" action="/signup">
        <label class="form-control-label" for="userName">
           User Name</label>
        {{with .Errors.username}}
        <div class="alert alert-danger">
            {{.}}
        </div>
        {{end}}
        <input class="form-control" type="text" id="
           userName" name="userName">
        <label class="form-control-label" for="fName">
           First Name</label>
        {{with .Errors.fname}}
        <div class="alert alert-danger">
            {{.}}
        </div>
        {{end}}
        <input class="form-control" type="text" id="fName
           " name="fName">
        <label class="form-control-label" for="lName">
           Last Name</label>
        {{with .Errors.lname}}
        <div class="alert alert-danger">
            {{.}}
        </div>
        {{end}}
        <input class="form-control" type="text" id="lName
           " name="lName">
        <label class="form-control-label" for="email">
           Email</label>
        {{with .Errors.email}}
        <div class="alert alert-danger">
            {{.}}
        </div>
        {{end}}
        <input class="form-control" type="email" id="
           email" name="email">
        <label class="form-control-label" for="password">
           Password</label>
        {{with .Errors.password}}
        <div class="alert alert-danger">
            {{.}}
        </div>
        {{end}}
        <input class="form-control" type="password" id="
           password" name="password">
        <label class="form-control-label" for="cpassword"
```

```
            >Confirm Password</label>
42          <input class="form-control" type="password" id="
               cpassword" name="cpassword">
43          <button class="btn btn-primary" type="submit">
               Submit</button>
44      </form>
45  </div>
46  <div class="alert alert-info">
47      <a href="/" class="alert">Sign In!</a>
48  </div>
49  {{template "footer" .}}
50  {{end}}
```

```
{{define "internal"}}
{{template "header" .}}

<div class="navbar navbar-inverse navrbar-fixed-top" >
  <div class="container-fluid">
    <div class = "navbar-header">
      <a href="#" class="navbar-brand">137132696</a>
    </div>
    <ul class="nav navbar-nav">
    <!-- <li style="width:300px;left:10px;top:10px;">
       <input type="text" class="form-control" id="search
       "></li>  -->
    <li style="width:300px;left:10px;top:10px;"><input
       type="text" class="form-control" id="myInput"
       onkeyup="myFunction()" placeholder="Search for
       mobile devices.." title="Type in a name">
    <li style="top:10px;left:20px;"><button type="submit"
        class="btn btn-link navbar-btn navbar-link"><span
        class="glyphicon glyphicon-search" ></span></
       button></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
    <li><a href="#" class="dropdown-toggle" data-toggle="
       dropdown"><span class="glyphicon
       glyphicon-shopping-cart"></span>Cart <span id="
       count-cart" style="
       color:white;font-size:12;background-color:green;border-radius:
        100px;"> 0</span></a>
      <div class="dropdown-menu" style="width:800px;">
        <div class="panel panel-info">
          <div class="panel-heading" ></div>
            <div class="row" >
              <div class="col-md-2"><b>Product Image</b>
                 </div>
              <div class="col-md-2"><b>Product Name</b>
                 </div>
              <div class="col-md-2"><b>Quantity</b></div>
              <div class="col-md-2"><b>Price</b></div>
              <div class="col-md-2"><b>Item Total</b>
                 </div>
              <div class="col-md-2"><b>Add Remove</b>
                 </div>
            </div>
          <div class="panel-body">
              <div id="show-cart">
              <!-- -->
              </div>
              <p><br/></p>
              <form action="/placeorder" method="post">
              <button class="btn btn-danger btn-sm"
                 style="float:right;">Place Order</button
```

```
                    >
34                </form>
35              </div>
36              <div class="panel-footer"></div>
37            </div>
38          </div>
39      </li>
40      <li><a href="#" class="dropdown-toggle" data-toggle="
           dropdown"><span class="glyphicon glyphicon-user"><
           /span>Hi {{.Usr.Fnm}}!</a>
41          <ul class="dropdown-menu" style="width:160px;">
42            <li><a href="#" style="text-decoration:none;
                 color:blue;">Manage Profile</a></li>
43            <li class="divider"></li>
44            <li><a href="#" style="text-decoration:none;
                 color:blue;">Change Password</a></li>
45          </ul>
46      </li>
47      <li><form action="/logout" method="post"><button
           type="submit" class="btn btn-link navbar-btn
           navbar-link"><span class="glyphicon
           glyphicon-log-out"></span> Log off</button></form>
           </li>
48      <!-- <li><form action="/logout" method="post"><button
            class="glyphicon glyphicon-log-out" type="submit"
           >Logout</form></li> -->
49      </ul>
50    </div>
51  </div>
52
53  <!--
54  <div class="alert alert-info" role="alert">
55      <h1 class="alert">Welcome! {{.Usr.Usrname}} </h1>
56      <hr>
57
58      <strong>Hello User: </strong>
59  </div>
60  <div class="jumbotron">
61      <div class="container">
62          what would you like to do today {{.Usr.Fnm}} {{.
               Usr.Lnm}} ?
63      </div>
64  </div>
65  -->
66
67  <p><br/></p>
68  <p><br/></p>
69  <div class="container-fluid">
70    <div class="row">
71      <div class="col-md-1"></div>
72      <!-- <div class="col-md-2">
```

```
73        <div class="nav nav-pills nav-stacked">
74        <li class="active"><a href="#">Mobile Phones</a></
              li>
75        <li class="active"><a href="#"></a>Something</li>
76        </div>
77      </div>
78      -->
79      <div class="col-md-10">
80        <div id="show-alert">
81        <!-- -->
82        </div>
83        <div class="panel panel-info">
84          <div class="panel-heading">Products</div>
85          <div class="panel-body">
86            {{with .Prd}}
87            {{range .}}
88            <div class="col-md-4 a" >
89              <div class="panel panel-info b">
90                <div class="panel-heading c">{{.Pname}}
                      </div>
91                <div class="panel-body">
92                  <img src="/files/{{.Image}}" alt="HTML5
                        Icon" style="width:200px;height:200px;
                        ">
93                </div>
94                <div class="panel-heading">&#8377; {{.Price
                      }}
95                  <!-- <input type="number" min="1" style="
                        width:50px;margin-left:60px;border:1px
                         solid red;
                        border-radius:4px;background-color:white;
                        "></input> -->
96                  <button style="float:right;" class="btn
                        btn-danger btn-sm add-to-cart"
                        data-pid="{{.Puid}}" data-image="{{.
                        Image}}" data-name="{{.Pname}}"
                        data-price="{{.Price}}">AddtoCart</
                        button>
97                </div>
98              </div>
99            </div>
100           {{end}}
101           {{end}}
102         </div>
103         <div class="panel-footer">Product List</div>
              </div>
104       </div>
105     </div>
106     <div class="col-md-1"></div>
107   </div>
108 </div>
```

```
109
110
111 <script src="/initializr/js/shoppingCart.js"></script>
112
113 <script>
114
115 $(".add-to-cart").click(function(event){
116   event.preventDefault();
117   var pid = $(this).attr("data-pid");
118   var image = $(this).attr("data-image");
119   var name = $(this).attr("data-name");
120   var price = Number($(this).attr("data-price"));
121   shoppingCart.addItemToCart(pid, image, name, price, 1);
122   displayCart();
123   showAlert();
124 });
125
126 $("#clear-cart").click(function(event){
127   shoppingCart.clearCart();
128   displayCart();
129 });
130
131 function displayCart() {
132   var cartArray = shoppingCart.listCart();
133   var output = "";
134   for (var i in cartArray) {
135     output += "<div class='row' style='border-bottom: 1px
          solid #4588ba; margin-bottom:10px;'>"
136           +"<div class='col-md-2'>"+"<img src='/files/"
                +cartArray[i].image+"' alt='HTML5 Icon'
                style='width:50px;height:50px;'>"+"</div>"
137           +"<div class='col-md-2'>"+cartArray[i].name+"
                </div>"
138           +"<div class='col-md-2'>"
139           +" <input class='item-count' type='number'
                min='1' data-name='"+cartArray[i].name+"'
                value='"+cartArray[i].count+"' style='
                width:50px;border:1px solid red;
                border-radius:4px;background-color:white;'
                 ></input>"
140           +"</div>"
141           +"<div class='col-md-2'>"
142           +"&#8377; "+cartArray[i].price
143           +"</div>"
144           +"<div class='col-md-2'>"
145           +"&#8377; "+cartArray[i].total
146           +"</div>"
147           +"<div class='col-md-2'>"
148           +" <button class='plus-item' data-name='"+
                cartArray[i].name+"' style='
                background-color:#4CAF50; border:none;
```

```
                    color:white;font-size: 16px'>+</button>"
149                 +" <button class='subtract-item' data-name='"
                    +cartArray[i].name+"' style='
                    background-color:#f44336; border:none;
                    color:white;font-size: 16px'>-</button>"
150                 +" <button class='delete-item' data-name='"+
                    cartArray[i].name+"' style='
                    background-color:white; border:none;
                    color:#f44336;font-size: 16px'><b>X</b></
                    button>"
151                 +"</div>"
152               +"</div>";
153     }
154     $("#show-cart").html(output);
155     $("#count-cart").html( shoppingCart.countCart() );
156     $("#total-cart").html( shoppingCart.totalCart() );
157 }
158
159 $("#show-cart").on("click",".plus-item",function(event){
160     event.preventDefault();
161     var pid = $(this).attr("data-pid");
162     var image = $(this).attr("data-image");
163     var name = $(this).attr("data-name");
164     shoppingCart.addItemToCart(pid, image, name, 0, 1);
165     displayCart();
166 });
167
168 $("#show-cart").on("click",".subtract-item",function(
      event){
169     event.preventDefault();
170     var name = $(this).attr("data-name");
171     shoppingCart.removeItemFromCart(name);
172     displayCart();
173 });
174
175 $("#show-cart").on("click",".delete-item",function(event)
      {
176     event.preventDefault();
177     var name = $(this).attr("data-name");
178     shoppingCart.removeItemFromCartAll(name);
179     displayCart();
180 });
181
182 $("#show-cart").on("change",".item-count",function(event)
      {
183     event.preventDefault();
184     var name = $(this).attr("data-name");
185     var count = Number( $(this).val() );
186     shoppingCart.setCountForItem(name, count);
187     displayCart();
188 });
```

```
189
190 function showAlert() {
191   alert = "<div class='alert alert-success
          alert-dismissable'>"
192       +"<a href='#' class='close' data-dismiss='alert'
             aria-label='close'>&times;</a>"
193       +"Product has been added to the cart sucessfully!!"
194       +"</div>"
195   $("#show-alert").html(alert);
196 }
197
198 function myFunction() {
199   var input, filter, panel;
200   input = document.getElementById("myInput");
201   filter = input.value.toUpperCase();
202   a = document.getElementsByClassName("a");
203   // console.log(a);
204   // Loop through all table rows, and hide those who don'
          t match the search query
205
206   if (a) {
207   for (i = 0; i < a.length; i++) {
208     b = a[i].getElementsByClassName("b");
209     // console.log(b);
210     if(b) {
211       for (j = 0; j < b.length; j++) {
212         c = b[j].getElementsByClassName("c");
213         console.log(c);
214         if (c) {
215           for (k = 0; k < c.length; k++) {
216             d = c[k];
217             if (d.innerHTML.toUpperCase().indexOf(filter)
                    > -1) {
218               a[i].style.display = "";
219             } else {
220               a[i].style.display = "none";
221             }
222           }
223         }
224       }
225     }
226   }
227   }
228  }
229
230 </script>
231
232 {{template "footer" .}}
233 {{end}}
```

```
{{define "Placeorder"}}
{{template "header" .}}

<!-- <textarea id="Output"></textarea> -->
<div class="navbar navbar-light navrbar-fixed-top" style=
   "background-color: #e3f2fd;">
  <div class="container-fluid">
    <div class = "navbar-header">
      <a href="#" class="navbar-brand">137132696</a>
    </div>
    <ul class="nav navbar-nav"></ul>
    <ul class="nav navbar-nav navbar-right">
    <li><form><button type="submit" class="btn btn-link
       navbar-btn navbar-link" onclick="goBack()"><span
       class="glyphicon glyphicon-hand-left"></span>
       Back</button></form></li>
    <li><form action="/logout" method="post"><button
       type="submit" class="btn btn-link navbar-btn
       navbar-link"><span class="glyphicon
       glyphicon-log-out"></span> Log off</button></form>
       </li>
    </ul>
    </div>
  </div>
</div>
<div class="container">
    <div class="panel panel-info">
      <div class="panel-heading"></div>
        <div class="row">
          <div class="col-md-3"><b>Product Image</b></div
             >
          <div class="col-md-3"><b>Product Name</b></div>
          <div class="col-md-2"><b>Quantity</b></div>
          <div class="col-md-2"><b>Price</b></div>
          <div class="col-md-2"><b>Item Total</b></div>
        </div>
          <div class="panel-body">
              <div id="show-cart">
              <!-- -->
              </div>
              <p><br/></p>
              <div style="float:right;">
              <form action="/payment" method="POST">
              <input id="Output" type="hidden" name="
                 Output">
              <input id="tot-count" type="hidden" name="
                 totalcount">
              <input id="tot-price" type="hidden" name="
                 totalprice">
                <script
                  src="https://checkout.stripe.com/
```

```
                    checkout.js" class="stripe-button"
                    style="float:right;"
40                  data-key={{.}}
41                  data-currency ="INR"
42                  data-amount= ""
43                  data-name="Payment Gateway"
44                  data-description="Online Mobile Store"
45                  data-image="https://stripe.com/img/
                    documentation/checkout/
                    marketplace.png"
46                  data-locale="auto"
47                  data-shipping-address="true">
48                </script>
49              </form>
50              </div>
51          </div>
52          <div class="panel-footer"></div>
53      </div>
54  </div>
55
56  <script>
57  $('#Output').val(localStorage.getItem('shoppingCart'));
58
59  function displayTable() {
60    var tableArray = JSON.parse(localStorage.getItem("
        shoppingCart"));
61    var tot_count = 0;
62    var tot_price = 0;
63    for (var j in tableArray) {
64      tableArray[j].total = (tableArray[j].count *
          tableArray[j].price).toFixed(2);
65      tot_count += tableArray[j].count;
66      tot_price += (Number(tableArray[j].total));
67    }
68    var output = "";
69    for (var i in tableArray) {
70      output += "<div class='row' style='border-bottom: 1px
          solid #4588ba; margin-bottom:10px;'>"
71            +"<div class='col-md-3'>"+"<img src='/files/"
                +tableArray[i].image+"' alt='HTML5 Icon'
                style='width:100px;height:100px;'>"+"</div
                >"
72            +"<div class='col-md-3'>"+tableArray[i].name+
                "</div>"
73            +"<div class='col-md-2'>"+tableArray[i].count
                +"</div>"
74            +"<div class='col-md-2'>"+" &#8377; "+
                tableArray[i].price+"</div>"
75            +"<div class='col-md-2'>"+" &#8377; "+
                tableArray[i].total+"</div>"
76          +"</div>";
```

```
77    }
78    output += "<div class='row' style='border-bottom: 1px
          solid #4588ba; margin-bottom:10px;'>"
79              +"<div class='col-md-3'>"+""+"</div>"
80              +"<div class='col-md-3'>"+""+"</div>"
81              +"<div class='col-md-2'>"+"<b>"+tot_count+"</
                    b></div>"
82              +"<div class='col-md-2'>"+""+"</div>"
83              +"<div class='col-md-2'>"+"<b>"+" &#8377; "+
                    tot_price.toFixed(2)+"</b></div>"
84          +"</div>";
85
86    $("#show-cart").html(output);
87    $("#tot-count").val(tot_count);
88    $("#tot-price").val(tot_price * 100);
89
90  }
91
92  displayTable();
93
94  function goBack() {
95      window.history.back();
96  }
97
98  </script>
99
100 {{template "footer" .}}
101 {{end}}
```

```
{{define "Payment"}}
{{template "header" .}}

<!-- <div id="load_screen" style="position:absolute;
    width:100%; text-align:center;"><div id="loading">
    <image src="/files/ajax-loader.gif" border="0"></image
    > <br> Please wait while we process your request....
    </div></div> -->

<div class="navbar navbar-light navrbar-fixed-top" style=
    "background-color: #e3f2fd;">
  <div class="container-fluid">
    <div class = "navbar-header">
      <a href="#" class="navbar-brand">137132696</a>
    </div>
    <ul class="nav navbar-nav"></ul>
    <ul class="nav navbar-nav navbar-right">
    <li><form action="/example" method="post"><button
       type="submit" class="btn btn-link navbar-btn
       navbar-link"><span class="glyphicon glyphicon-home
       "></span> Home</button></form></li>
    <li><form action="/logout" method="post"><button
       type="submit" class="btn btn-link navbar-btn
       navbar-link"><span class="glyphicon
       glyphicon-log-out"></span> Log off</button></form>
       </li>
    </ul>
    </div>
  </div>
</div>


<div class="container">
  <div class="col-md-1"></div>
  <div class="col-md-10">
    <div class="panel panel-info">
      <div class="panel-heading"><b>Order Invoice &#47;
         Payment Receipt</b></div>
       <div class="panel-body">
         <div class="row">
           <div class="col-md-3"><b>Order ID:</b></div>
           <div class="col-md-7">{{.ouid}}</div>
         </div>
         <div class="row">
           <div class="col-md-3"> </div>
           <div class="col-md-7"> </div>
         </div>
         <div class="row">
           <div class="col-md-3"><b>Shipping Address</b>
             </div>
           <div class="col-md-7">{{.
```

```
            shipdetail.shaddressname}}</div>
38          </div>
39          <div class="row">
40            <div class="col-md-3"></div>
41            <div class="col-md-7">{{.
                shipdetail.shaddressline}}</div>
42          </div>
43          <div class="row">
44            <div class="col-md-3"></div>
45            <div class="col-md-7">{{.
                shipdetail.shaddresscity}}</div>
46          </div>
47          <div class="row">
48            <div class="col-md-3"></div>
49            <div class="col-md-7">{{.
                shipdetail.shaddresszip}}</div>
50          </div>
51          <div class="row">
52            <div class="col-md-3"></div>
53            <div class="col-md-7">{{.
                shipdetail.shaddresscountry}}</div>
54          </div>
55          <div class="row">
56            <div class="col-md-3"> </div>
57            <div class="col-md-7"> </div>
58          </div>
59          <div class="row">
60            <div class="col-md-3"><b>Email Address:</b>
                </div>
61            <div class="col-md-7">{{.shipdetail.eladdress
                }}</div>
62          </div>
63          <div class="row">
64            <div class="col-md-3"> </div>
65            <div class="col-md-7"> </div>
66          </div>
67          <div class="row">
68            <div class="col-md-3"><b>Ordered Items:</b>
                </div>
69            <div class="col-md-7"></div>
70          </div>
71          <div class="row">
72            <div class="col-md-3"> </div>
73            <div class="col-md-7"> </div>
74          </div>
75          {{ with .orderdetail}}
76          {{range .}}
77          <div class="row">
78            <div class="col-md-3"><img src="/files/{{.
                image}}" style="width:50px;height:50px;">
                </div>
```

```
79              <div class="col-md-7"> </div>
80                <div class="col-md-3"><b>Product: </b>{{.
                     name}} </div>
81                <div class="col-md-2"><b>Qty: </b>{{.count
                     }}</div>
82                <div class="col-md-2"><b>Price: </b>&#8377;
                      {{.price}}</div>
83            </div>
84            {{end}}
85            {{end}}
86            <div class="row">
87              <div class="col-md-3"> </div>
88              <div class="col-md-7"> </div>
89            </div>
90            <div class="row">
91              <div class="col-md-3"><b>Total Amount:</b>
                    </div>
92              <div class="col-md-7">&#8377; {{.
                    chargedamount}}</div>
93            </div>
94            <div class="row">
95              <div class="col-md-3"> </div>
96              <div class="col-md-7"> </div>
97            </div>
98            <div class="row">
99              <div class="col-md-3"><b>Order DateTime:</b>
                    </div>
100             <div class="col-md-7">{{.timestamp}}</div>
101           </div>
102         </div>
103         <div class="panel-footer"></div>
104     </div>
105     </div>
106   <div class="col-md-1"></div>
107 </div>
108
109
110 <!--
111 <script>
112     window.addEventListener("load", function() {
113         var load_screen = document.getElementById("
                load_screen");
114         document.body.removeChild(load_screen);
115       }
116     );
117 </script>
118 -->
119
120 {{template "footer" .}}
121 {{end}}
```

```
{{define "Admin"}}
{{template "header" .}}

<div class="navbar navbar-inverse navrbar-fixed-top" >
  <div class="container-fluid">
    <div class = "navbar-header">
      <a href="#" class="navbar-brand">137132696</a>
    </div>
    <ul class="nav navbar-nav">
      <li><form action="/manageproduct" method="post">
          <button type="submit" class="btn btn-link
          navbar-btn navbar-link"><span class="glyphicon
          glyphicon-th-list"></span> Manage Products</
          button></form></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
    <li><button type="submit" class="btn btn-link
       navbar-btn navbar-link"><span class="glyphicon
       glyphicon-user"></span> Hi Admin!</button></li>
    <li><form action="/logout" method="post"><button
       type="submit" class="btn btn-link navbar-btn
       navbar-link"><span class="glyphicon
       glyphicon-log-out"></span> Log off</button></form>
       </li>
    </ul>
  </div>
</div>

<div class="container">
  <div class="col-md-1"></div>
  <div class="col-md-10">
    <div class="panel panel-info">
      <div class="panel-heading">Order Data</div>
        <div class="panel-body">
          <table style="width:100%">
            <tr style="border-bottom: 1px solid #ddd;">
              <th>OrderID</th>
              <th>Order Amount</th>
              <th>Timestamp</th>
            </tr>
            {{ range . }}
            <tr style="border-bottom: 1px solid #ddd;">
              <!-- <td><a href="#">{{.ouid}}</a></td> -->
              <td><form action="/vieworder" method="post"
                 ><button class="btn btn-link" name="
                 orderid" value="{{.ouid}}" style="
                 padding-top:20px;">{{.ouid}}</button></
                 form></td>
              <td >&#8377; {{printf "%.2f" .chargedamount
                 }}</td>
              <td>{{.timestamp}}</td>
```

```
37              </tr>
38              {{end}}
39            </table>
40          </div>
41          <div class="panel-footer"></div>
42      </div>
43      </div>
44    <div class="col-md-1"></div>
45  </div>
46
47
48  {{template "footer" .}}
49  {{end}}
```

```
{{define "Vieworder"}}
{{template "header" .}}

<div class="navbar navbar-light navrbar-fixed-top" style=
    "background-color: #e3f2fd;">
  <div class="container-fluid">
    <div class = "navbar-header">
      <a href="#" class="navbar-brand">137132696</a>
    </div>
    <ul class="nav navbar-nav">
      <li><form action="/admin" method="post"><button
          type="submit" class="btn btn-link navbar-btn
          navbar-link"><span class="glyphicon
          glyphicon-home"></span> Home</button></form></li
          >
    </ul>
    <ul class="nav navbar-nav navbar-right">
    <li><button type="submit" class="btn btn-link
        navbar-btn navbar-link"><span class="glyphicon
        glyphicon-user"></span> Hi Admin!</button></li>
    <li><form action="/logout" method="post"><button
        type="submit" class="btn btn-link navbar-btn
        navbar-link"><span class="glyphicon
        glyphicon-log-out"></span> Log off</button></form>
        </li>
    </ul>
  </div>
</div>

<div class="container">
  <div class="col-md-1"></div>
  <div class="col-md-10">
    <div class="panel panel-info">
      <div class="panel-heading"><b>Order Details</b>
         </div>
        <div class="panel-body">
          <div class="row">
            <div class="col-md-3"><b>Order ID:</b></div>
            <div class="col-md-7">{{.ouid}}</div>
          </div>
          <div class="row">
            <div class="col-md-3"> </div>
            <div class="col-md-7"> </div>
          </div>
          <div class="row">
            <div class="col-md-3"><b>Shipping Address</b>
               </div>
            <div class="col-md-7">{{.
               shipdetail.shaddressname}}</div>
          </div>
          <div class="row">
```

```
38              <div class="col-md-3"></div>
39              <div class="col-md-7">{{.
                    shipdetail.shaddressline}}</div>
40            </div>
41            <div class="row">
42              <div class="col-md-3"></div>
43              <div class="col-md-7">{{.
                    shipdetail.shaddresscity}}</div>
44            </div>
45            <div class="row">
46              <div class="col-md-3"></div>
47              <div class="col-md-7">{{.
                    shipdetail.shaddresszip}}</div>
48            </div>
49            <div class="row">
50              <div class="col-md-3"></div>
51              <div class="col-md-7">{{.
                    shipdetail.shaddresscountry}}</div>
52            </div>
53            <div class="row">
54              <div class="col-md-3"> </div>
55              <div class="col-md-7"> </div>
56            </div>
57            <div class="row">
58              <div class="col-md-3"><b>Email Address:</b>
                    </div>
59              <div class="col-md-7">{{.shipdetail.eladdress
                    }}</div>
60            </div>
61            <div class="row">
62              <div class="col-md-3"> </div>
63              <div class="col-md-7"> </div>
64            </div>
65            <div class="row">
66              <div class="col-md-3"><b>Ordered Items:</b>
                    </div>
67              <div class="col-md-7"></div>
68            </div>
69            <div class="row">
70              <div class="col-md-3"> </div>
71              <div class="col-md-7"> </div>
72            </div>
73            {{ with .orderdetail}}
74            {{range .}}
75            <div class="row">
76              <div class="col-md-3"><img src="/files/{{.
                    image}}" style="width:50px;height:50px;">
                    </div>
77              <div class="col-md-7"> </div>
78                <div class="col-md-3"><b>Product: </b>{{.
                      name}} </div>
```

```
79                  <div class="col-md-2"><b>Qty: </b>{{.count
                        }}</div>
80                  <div class="col-md-2"><b>Price: </b>&#8377;
                        {{.price}}</div>
81              </div>
82              {{end}}
83              {{end}}
84              <div class="row">
85                <div class="col-md-3"> </div>
86                <div class="col-md-7"> </div>
87              </div>
88              <div class="row">
89                <div class="col-md-3"><b>Total Amount:</b>
                      </div>
90                <div class="col-md-7">&#8377; {{.
                      chargedamount}}</div>
91              </div>
92              <div class="row">
93                <div class="col-md-3"> </div>
94                <div class="col-md-7"> </div>
95              </div>
96              <div class="row">
97                <div class="col-md-3"><b>Order DateTime:</b>
                      </div>
98                <div class="col-md-7">{{.timestamp}}</div>
99              </div>
100           </div>
101           <div class="panel-footer"></div>
102     </div>
103     </div>
104   <div class="col-md-1"></div>
105 </div>
106
107 {{template "footer" .}}
108 {{end}}
```

```
{{define "Manageproduct"}}
{{template "header" .}}

<div class="navbar navbar-inverse navrbar-fixed-top" >
  <div class="container-fluid">
    <div class = "navbar-header">
      <a href="#" class="navbar-brand">137132696</a>
    </div>
    <ul class="nav navbar-nav">
      <li><form action="/admin" method="post"><button
         type="submit" class="btn btn-link navbar-btn
         navbar-link"><span class="glyphicon
         glyphicon-th-list"></span> View Orders</button><
         /form></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
    <li><button type="submit" class="btn btn-link
       navbar-btn navbar-link"><span class="glyphicon
       glyphicon-user"></span> Hi Admin!</button></li>
    <li><form action="/logout" method="post"><button
       type="submit" class="btn btn-link navbar-btn
       navbar-link"><span class="glyphicon
       glyphicon-log-out"></span> Log off</button></form>
       </li>
    </ul>
  </div>
</div>
<div class="container">
  <div class="col-md-1"></div>
  <div class="col-md-10">
    <div class="panel panel-primary">
      <div class="panel-heading"><b>Add Product</b></div>
      <div class="panel-body">
        <form action="/addproduct" method="get">
          <h2 style="font-size:16"> Click on the button
             to add a new product</h2>
          <button class="btn btn-warning" type="submit"
             style="float:right;">Add Product</button>
        </form>
      </div>
      <div class="panel-footer"></div>
    </div>
    <div class="panel panel-info">
      <div class="panel-heading"><b>Manage Products</b>
         </div>
      <div class="panel-body">
        <table style="width:100%;">
            <tr style="border-bottom: 1px solid #ddd;">
              <th>Product Image</th>
              <th>Product Name</th>
              <th>Product Quantity</th>
```

```
39                <th>Product Price</th>
40                <th>Delete </th>
41                <th>Update</th>
42              </tr>
43              {{range $index, $value := .}}
44
45              <tr style="border-bottom: 1px solid #ddd;">
46                <td><img src="/files/{{.image}}" style="
                     width:100px;height:100px;"></td>
47                <td>{{.pname}}</td>
48                <td>{{.quantity}}</td>
49                <td>&#8377; {{.price}}</td>
50                <td ><form action="/deleteproduct" method="
                     get"><button class="btn btn-danger"
                     name="proid" value="{{.puid}}" style="
                     border:none;margin-right:10px;">Delete <
                     /button></form></td>
51                <td ><form onsubmit="save_data(this, {{
                     $index}})" action="/showproduct" method=
                     "get"><button class="btn btn-info prodid
                     " id="pid{{$index}}"  value="{{
                     $value.puid}}" style="
                     border:none;margin-right:10px;">Update</
                     button></form></td>
52              </tr>
53
54              {{end}}
55
56          </table>
57        </div>
58        <div class="panel-footer"></div>
59      </div>
60    </div>
61    <div class="col-md-1"></div>
62  </div>
63
64  <script>
65
66  function save_data(form, rowno) {
67    var input = document.getElementById("pid"+rowno);
68    localStorage.setItem("prid", input.value);
69    document.getElementById("pid"+rowno).name = 'prodid';
70    // var storedValue = localStorage.getItem("prid");
71  }
72
73  </script>
74
75  {{template "footer" .}}
76  {{end}}
```

```
{{define "Addproduct"}}
{{template "header" .}}

<div class="navbar navbar-light navbar-fixed-top" style="
    background-color: #e3f2fd;" >
  <div class="container-fluid">
    <div class = "navbar-header">
      <a href="#" class="navbar-brand">137132696</a>
    </div>
    <ul class="nav navbar-nav">
      <li><form action="/manageproduct" method="post">
          <button type="submit" class="btn btn-link
          navbar-btn navbar-link"><span class="glyphicon
          glyphicon-th-list"></span> Manage Products</
          button></form></li>
      <li><form action="/admin" method="post"><button
          type="submit" class="btn btn-link navbar-btn
          navbar-link"><span class="glyphicon
          glyphicon-th-list"></span> View Orders</button><
          /form></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
    <li><form action="/admin" method="post"><button type=
        "submit" class="btn btn-link navbar-btn
        navbar-link"><span class="glyphicon
        glyphicon-hand-left"></span> Back</button></form><
        /li>
    <li><form action="/logout" method="post"><button
        type="submit" class="btn btn-link navbar-btn
        navbar-link"><span class="glyphicon
        glyphicon-log-out"></span> Log off</button></form>
        </li>
    </ul>
  </div>
</div>

<p><br/></p>
<p><br/></p>
<p><br/></p>


<div class="container-fluid">
  <div class="row">
    <div class="col-md-1"></div>
    <div class="col-md-10">
      <div class="panel panel-info">
        <div class="panel-heading">Add Product Form</div>
          <div class="container-fluid" style="
              margin-top:20px;">
            <form  method="POST" action="/addproduct"
              enctype="multipart/form-data" runat="
```

```
 server">
<label class="form-control-label" for="
   productName">Product Name</label>
{{with .Errors.pname}}
<div class="alert alert-danger">
  {{.}}
</div>
{{end}}
<input class="form-control" type="text" id=
   "productName" name="productName">
<label class="form-control-label" for="
   quantity">Quantity</label>
{{with .Errors.quantity}}
<div class="alert alert-danger">
  {{.}}
</div>
{{end}}
<input class="form-control" type="text" id=
   "quantity" name="quantity">
<label class="form-control-label" for="
   price">Price</label>
{{with .Errors.price}}
<div class="alert alert-danger">
  {{.}}
</div>
{{end}}
<input class="form-control" type="text" id=
   "price" name="price">
<div class = "jumbotron">
  <!-- <h1 class = "test-center" style="
     font-size:40;color:gray;"> Upload
     Image of the Product</h1> -->
  <img id="pic" src="/files/
     uploaddefault1.jpeg"  alt="Upload
     Image of the Product" style="width:100
     %;font-size:40;color:gray;" />
</div>
<label class="form-control-label" for="
   productimage"></label>
{{with .Errors.image}}
<div class="alert alert-danger">
  {{.}}
</div>
{{end}}

<!-- <input type="file" name="productimage"
    id = "productimage" multiple="multiple"
    class = "btn btn-danger" style="
   margin-bottom:20px;"> -->

<input type="file" name="productimage" id =
```

```
                    "productimage" multiple="multiple"
                   class = "btn btn-danger">

                <input type="submit" name="submit" value="
                   Submit" class = "btn btn-info">

            </form>
          </div>

        <div class="panel footer"></div>
      </div>
    </div>
    <div class="col-md-1"></div>
  </div>
</div>



<!--
<h1 class="alert alert-info">Add Product Form</h1>
<div class="container-fluid">
    <form  method="POST" action="/addproduct" enctype="
       multipart/form-data" runat="server">
        <label class="form-control-label" for="
           productName">Product Name</label>
        {{with .Errors.pname}}
        <div class="alert alert-danger">
            {{.}}
        </div>
        {{end}}
        <input class="form-control" type="text" id="
           productName" name="productName">
        <label class="form-control-label" for="quantity">
           Quantity</label>
        {{with .Errors.quantity}}
        <div class="alert alert-danger">
            {{.}}
        </div>
        {{end}}
        <input class="form-control" type="text" id="
           quantity" name="quantity">
        <label class="form-control-label" for="price">
           Price</label>
        {{with .Errors.price}}
        <div class="alert alert-danger">
            {{.}}
        </div>
        {{end}}
        <input class="form-control" type="text" id="price
           " name="price">
    <div class = "jumbotron">
```

```
109         <!--  <<h1 class = "test-center"> Upload Image of
                the Product</h1>
110         <img id="pic" src="/files/uploaddefault1.jpeg"
                alt="Upload Image of the Product" style="
                font-size:40;color:gray;" />
111     </div>
112         <label class="form-control-label" for="
                productimage"></label>
113         {{with .Errors.image}}
114         <div class="alert alert-danger">
115             {{.}}
116         </div>
117         {{end}}
118
119
120     <!-- <input type="file" name="productimage" id = "
            productimage" multiple="multiple" class = "btn
            btn-danger">
121
122     <input type="file" name="productimage" id = "
            productimage" multiple="multiple" class = "btn
            btn-danger">
123
124
125         <input type="submit" name="submit" value="Submit"
                class = "btn btn-info">
126
127
128     </form>
129 </div>
130 -->
131
132 <!--
133 <div class="container-fluid">
134 <form action="/admin" method="post">
135     <button class="btn btn-warning" type="submit">Back to
            Admin</button>
136 </form>
137 </div>
138 -->
139
140
141
142 <script>
143 function readURL(input) {
144     if (input.files && input.files[0]) {
145         var reader = new FileReader();
146
147         reader.onload = function (e) {
148             $('#pic').attr('src', e.target.result);
149         }
```

```
150
151             reader.readAsDataURL(input.files[0]);
152         }
153     }
154
155     $("#productimage").change(function(){
156         readURL(this);
157     });
158     </script>
159
160
161
162
163     {{template "footer" .}}
164     {{end}}
```

```
{{define "Showproduct"}}
{{template "header" .}}

<div class="navbar navbar-light navbar-fixed-top" style="
    background-color: #e3f2fd;" >
  <div class="container-fluid">
    <div class = "navbar-header">
      <a href="#" class="navbar-brand">137132696</a>
    </div>
    <ul class="nav navbar-nav">
      <li><form action="/manageproduct" method="post">
          <button type="submit" class="btn btn-link
          navbar-btn navbar-link"><span class="glyphicon
          glyphicon-th-list"></span> Manage Products</
          button></form></li>
      <li><form action="/admin" method="post"><button
          type="submit" class="btn btn-link navbar-btn
          navbar-link"><span class="glyphicon
          glyphicon-th-list"></span> View Orders</button><
          /form></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
    <li><form action="/admin" method="post"><button type=
        "submit" class="btn btn-link navbar-btn
        navbar-link"><span class="glyphicon
        glyphicon-hand-left"></span> Back</button></form><
        /li>
    <li><form action="/logout" method="post"><button
        type="submit" class="btn btn-link navbar-btn
        navbar-link"><span class="glyphicon
        glyphicon-log-out"></span> Log off</button></form>
        </li>
    </ul>
  </div>
</div>

<p><br/></p>
<p><br/></p>
<p><br/></p>



<div class="container-fluid">
  <div class="row">
    <div class="col-md-1"></div>
    <div class="col-md-10">
      <div class="panel panel-info">
        <div class="panel-heading">Update Product Form
          </div>
          <div class="container-fluid" style="
            margin-top:20px;">
```

```
33              <form  method="POST" action="/showproduct"
                   enctype="multipart/form-data" runat="
                   server">
34                <label class="form-control-label" for="
                     productName" >Product Name</label>
35                {{with .Errors.pname}}
36                <div class="alert alert-danger">
37                  {{.}}
38                </div>
39                {{end}}
40                <input class="form-control" type="text" id=
                     "productName" name="productName" value="
                     {{.Pname}}">
41                <label class="form-control-label" for="
                     quantity" >Quantity</label>
42                {{with .Errors.quantity}}
43                <div class="alert alert-danger">
44                  {{.}}
45                </div>
46                {{end}}
47                <input class="form-control" type="text" id=
                     "quantity" name="quantity" value="{{.
                     Quantity}}"  >
48                <label class="form-control-label" for="
                     price"  >Price</label>
49                {{with .Errors.price}}
50                <div class="alert alert-danger">
51                  {{.}}
52                </div>
53                {{end}}
54                <input class="form-control" type="text" id=
                     "price" name="price" value="{{.Price}}">
55                <div class = "jumbotron">
56                  <!-- <h1 class = "test-center" style="
                       font-size:40;color:gray;"> Upload
                       Image of the Product</h1> -->
57                  <img name="pic" id="pic" src="/files/{{.
                       Image}}"  alt="Upload Image of the
                       Product" style="width:100%;
                       font-size:40;color:gray;"  />
58                  <input type="hidden" name ="imgname" id="
                       imgname" value="{{.Image}}">
59                </div>
60                <label class="form-control-label" for="
                     productimage"></label>
61                {{with .Errors.image}}
62                <div class="alert alert-danger">
63                  {{.}}
64                </div>
65                {{end}}
66
```

```
67                  <!-- <input type="file" name="productimage"
                         id = "productimage" multiple="multiple"
                         class = "btn btn-danger" style="
                        margin-bottom:20px;"> -->
68
69                  <input type="file" name="productimage" id =
                         "productimage" multiple="multiple"
                        class = "btn btn-danger" value="{{.Image
                        }}" >
70
71                  <input type="submit" name="submit" value="
                        Submit" class = "btn btn-info">
72
73              </form>
74            </div>
75
76          <div class="panel footer"></div>
77        </div>
78      </div>
79      <div class="col-md-1"></div>
80    </div>
81  </div>
82
83
84
85  <script>
86  function readURL(input) {
87      if (input.files && input.files[0]) {
88          var reader = new FileReader();
89
90          reader.onload = function (e) {
91              $('#pic').attr('src', e.target.result);
92          }
93
94          reader.readAsDataURL(input.files[0]);
95      }
96  }
97
98  $("#productimage").change(function(){
99      readURL(this);
100 });
101 </script>
102
103
104 {{template "footer" .}}
105 {{end}}
```

```
{{define "Showproduct"}}
{{template "header" .}}

<div class="navbar navbar-light navbar-fixed-top" style="
    background-color: #e3f2fd;" >
  <div class="container-fluid">
    <div class = "navbar-header">
      <a href="#" class="navbar-brand">137132696</a>
    </div>
    <ul class="nav navbar-nav">
      <li><form action="/manageproduct" method="post">
          <button type="submit" class="btn btn-link
          navbar-btn navbar-link"><span class="glyphicon
          glyphicon-th-list"></span> Manage Products</
          button></form></li>
      <li><form action="/admin" method="post"><button
          type="submit" class="btn btn-link navbar-btn
          navbar-link"><span class="glyphicon
          glyphicon-th-list"></span> View Orders</button><
          /form></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
    <li><form action="/admin" method="post"><button type=
        "submit" class="btn btn-link navbar-btn
        navbar-link"><span class="glyphicon
        glyphicon-hand-left"></span> Back</button></form><
        /li>
    <li><form action="/logout" method="post"><button
        type="submit" class="btn btn-link navbar-btn
        navbar-link"><span class="glyphicon
        glyphicon-log-out"></span> Log off</button></form>
        </li>
    </ul>
  </div>
</div>

<p><br/></p>
<p><br/></p>
<p><br/></p>



<div class="container-fluid">
  <div class="row">
    <div class="col-md-1"></div>
    <div class="col-md-10">
      <div class="panel panel-info">
        <div class="panel-heading">Update Product Form
          </div>
          <div class="container-fluid" style="
            margin-top:20px;">
```

```
33              <form  onsubmit="load_data()" method="POST"
                   action="/showproduct" enctype="multipart/
                   form-data" runat="server">
34                <input type="hidden" name="prodid" id="
                     prodid" >
35                <label class="form-control-label" for="
                     productName" >Product Name</label>
36                {{with .Errors.pname}}
37                <div class="alert alert-danger">
38                  {{.}}
39                </div>
40                {{end}}
41                <input class="form-control" type="text" id=
                     "productName" name="productName" value="
                     {{.Pname}}">
42                <label class="form-control-label" for="
                     quantity" >Quantity</label>
43                {{with .Errors.quantity}}
44                <div class="alert alert-danger">
45                  {{.}}
46                </div>
47                {{end}}
48                <input class="form-control" type="text" id=
                     "quantity" name="quantity" value="{{.
                     Quantity}}"  >
49                <label class="form-control-label" for="
                     price"  >Price</label>
50                {{with .Errors.price}}
51                <div class="alert alert-danger">
52                  {{.}}
53                </div>
54                {{end}}
55                <input class="form-control" type="text" id=
                     "price" name="price" value="{{.Price}}">
56                <div class = "jumbotron">
57                  <!-- <h1 class = "test-center" style="
                       font-size:40;color:gray;"> Upload
                       Image of the Product</h1> -->
58                  <img name="pic" id="pic" src="/files/{{.
                       Image}}"  alt="Upload Image of the
                       Product" style="
                       width:400px;font-size:40;color:gray;"
                        />
59                  <input type="hidden" name ="imgname" id="
                       imgname" value="{{.Image}}">
60                </div>
61                <label class="form-control-label" for="
                     productimage"></label>
62                {{with .Errors.image}}
63                <div class="alert alert-danger">
64                  {{.}}
```

```
                </div>
                {{end}}

                <!-- <input type="file" name="productimage"
                    id = "productimage" multiple="multiple"
                    class = "btn btn-danger" style="
                   margin-bottom:20px;"> -->

                <input type="file" name="productimage" id =
                    "productimage" multiple="multiple"
                   class = "btn btn-danger" value="{{.Image
                   }}" >

                <input type="submit" name="submit" value="
                   Submit" class = "btn btn-info">

            </form>
          </div>

        <div class="panel footer"></div>
      </div>
    </div>
    <div class="col-md-1"></div>
  </div>
</div>



<script>
function readURL(input) {
    if (input.files && input.files[0]) {
        var reader = new FileReader();

        reader.onload = function (e) {
            $('#pic').attr('src', e.target.result);
        }

        reader.readAsDataURL(input.files[0]);
    }
}

$("#productimage").change(function(){
    readURL(this);
});

//
function load_data() {
  document.getElementById("prodid").value =
     localStorage.getItem("prid");
}

```

```
</script>


{{template "footer" .}}
{{end}}
```

4.3 Javascript Code

```
// ***************************Shopping Cart ******************************
//  ***********************************************************************


var shoppingCart = (function(){
  // private methods and properties
  var cart = [];

  function Item(pid, image, name, price, count) {
  this.pid = pid
  this.image = image
  this.name = name
  this.price = price
  this.count = count
  }

  function saveCart() {
    localStorage.setItem("shoppingCart", JSON.stringify(
       cart));
  };


  function loadCart() {
    this.cart = JSON.parse(localStorage.getItem("
       shoppingCart"));
  };

  loadCart();

  // public methods and properties
  var obj = {};

  obj.addItemToCart = function(pid, image, name, price,
     count) {
    for (var i in cart){
      if (cart[i].name === name) {
        cart[i].count += count;
        saveCart();
        return;
      }
    }
    var item = new Item(pid, image, name, price, count);
    cart.push(item);
    saveCart();
  };

  obj.setCountForItem = function(name, count) {
```

```
    for (var i in cart) {
      if (cart[i].name === name) {
        cart[i].count = count;
        break;
      }
    }
    saveCart();
  };


  obj.removeItemFromCart = function(name) {
    for (var i in cart){
      if (cart[i].name === name) {
        cart[i].count--;
        if (cart[i].count === 0) {
          cart.splice(i,1);
        }
        break;
      }
    }
    saveCart();
  };


  obj.removeItemFromCartAll = function(name) {
    for (var i in cart) {
      if (cart[i].name === name) {
        cart.splice(i,1);
        break;
      }
    }
    saveCart();
  };


  obj.clearCart = function() {
    cart = [];
    saveCart();
  };


  obj.countCart = function() {
    var totalCount = 0;
    for ( var i in cart) {
        totalCount += cart[i].count;
    }
    return totalCount;
  };


  obj.totalCart = function() {
```

```
95      var totalCost = 0;
96      for ( var i in cart) {
97          totalCost += cart[i].price * cart[i].count;
98      }
99      return totalCost.toFixed(2);
100   };
101
102
103   obj.listCart = function() {
104     var cartCopy = [];
105     for (var i in cart) {
106       var item = cart[i];
107       var itemCopy = {};
108       for (var p in item) {
109         itemCopy[p] = item[p];
110       }
111     itemCopy.total = (item.price * item.count).toFixed(2)
          ;
112     cartCopy.push(itemCopy);
113     }
114     return cartCopy;
115   };
116
117   return obj;
118
119 })();
```

4.4 Test Cases and Test Results

Table 4.1: **Test Cases for Req. ID FR1**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Enter website URL	User should be directed to secure HTTPS connection. User should provide security certificate details. Pass
2.0	Click on 'Sign-up' link on the landing page	User should be directed to Sign-up form page. Pass

Table 4.2: **Test Cases for Req. ID FR2**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Enter blank value and click on submit	Error ''Please enter a valid username'' should be received Pass
2.0	Enter a username that already exists and click on submit	Error ''Username already exists'' should be received Pass

Table 4.3: **Test Cases for Req. ID FR3**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Enter blank value in first name field and click on submit	Error ''Please enter a valid first name'' should be received Pass

2.0	Enter blank value in last name field and click on submit	Error ‘‘Please enter a valid last name’’ should be received Pass
3.0	Enter blank value in email address field and click on submit	Error ‘‘Please enter a valid email address’’ should be received Pass
4.0	Enter a string that does not conform to the regex pattern for an email address in email address field and click on submit	Error ‘‘Please enter a valid email address’’ should be received Pass
5.0	Enter a email address that already exists and click on submit	Error ‘‘email address already exists’’ should be received

Table 4.4: **Test Cases for Req. ID FR4**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Enter a blank value in the password field	Error ‘‘Please enter a valid password’’ should be received Pass
2.0	Enter a blank value in the confirm password field	Error ‘‘Please confirm your password’’ should be received Pass
3.0	Enter a incorrect value in the confirm password field	Error ‘‘Entered value does not match password’’ should be received Pass

Table 4.5: **Test Cases for Req. ID FR5**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Enter invalid values in any of the fields	Error alerts should be received as described in the above test cases Pass
2.0	Enter values that are valid and click submit	Form details should be posted to the server and if accepted be committed to the database in the user table. Pass

Table 4.6: **Test Cases for Req. ID FR6**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	At the login screen disable HTTPS in the browser settings. Try to login by entering valid username and password	Error should be received. Pass

Table 4.7: **Test Cases for Req. ID FR7**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Leave the username field blank and try to login	Error ``Username or password field are empty'' should be received Pass

2.0	Leave the password field blank and try to login	Error ``Username or password field are empty'' should be received Pass
3.0	Enter incorrect username or password and try to login	Error ``Invalid username or password'' should be received Pass
4.0	Enter correct username and password and try to login	User should be able to login. New Session ID unique for the username should get created. Pass

Table 4.8: **Test Cases for Req. ID FR8**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Enter invalid values in any of the fields	Error alerts should be received as described in the above test cases Pass

Table 4.9: **Test Cases for Req. ID FR9**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	After login, user should be presented the product listings page. All listing should be presented in the main view.	All products in the product catalog are listed and include image, name and price of the product. Pass

Table 4.10: **Test Cases for Req. ID FR10**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Type a search string to match a product by its name in the search box.	All products with the name that match the description criteria shall be displayed Pass

Table 4.11: **Test Cases for Req. ID FR11**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Select any new product on the listings page and add it to the shopping cart by clicking on the button	Product should get added to the shopping cart list. Local storage object should be created. Pass
2.0	Select any product on the listings page that has already been added to the shopping cart and add it again by clicking on the button	Quantity of the listed Product should get incremented by 1 unit. Local storage object should modified accordingly. Pass

Table 4.12: **Test Cases for Req. ID FR12**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Add a product to the shopping cart	Display count of the shopping cart should increase by 1. Product details that include the image, quantity being added, price per unit and the total cost for the selected product should be displayed in the shopping cart dropdown list Pass

Table 4.13: **Test Cases for Req. ID FR13**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	In the shopping cart dropdown list, change the quantity by using the number input control	Minimum value to which the quantity can be reduced is 1 Total product cost should change accordingly Pass
2.0	In the shopping cart dropdown list, click on the '+' button in the last column	Total product quantity should increase by 1 unit Total product cost should change accordingly Pass

3.0	In the shopping cart dropdown list, click on the '-' button in the last column	Total product quantity should decrease by 1 unit Total product cost should change accordingly Pass

Table 4.14: **Test Cases for Req. ID FR14**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Add a product to the shopping cart and change its quantity to different values	Total cost cannot be zero. Total Cost = Quantity * Price per unit Pass

Table 4.15: **Test Cases for Req. ID FR15**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	In the shopping cart dropdown list, click on the 'X' button in the last column	Line item for the product in the shopping cart list should get removed Pass

Table 4.16: **Test Cases for Req. ID FR16**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	With no products added to the shopping cart click on the 'Place Order' or checkout button	Error should be received Pass

2.0	With products added to the shopping cart click on the 'Place Order' or checkout button	User should be navigated to the Place Order page Pass
3.0	With products added to the shopping cart, disconnect the Internet connection. Then reconnect and click on the 'Place Order' or checkout button	User should not be navigated to the Place Order page. User should get logged out with message for session expired. Pass

Table 4.17: **Test Cases for Req. ID FR17**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Add products to shopping cart and checkout	Products should be listed with details i.e. the image, quantity being ordered, price per unit, total cost of the product. Pass

Table 4.18: **Test Cases for Req. ID FR18**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Check Order basket product total cost	Total cost cannot be zero. Total Cost = Quantity * Price per unit Pass

Table 4.19: **Test Cases for Req. ID FR19**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Check Order basket overall product quantity	Total order quantity cannot be zero. Total order quantity = sum of individual product quantities Pass
2.0	Check Order basket product total order cost	Total order cost cannot be zero. Total order Cost = sum of individual order cost by products added Pass

Table 4.20: **Test Cases for Req. ID FR20**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Click on 'Pay with card' button	Clicking on button should create a call to the Stripe Payment Gateway API and present the Stripe Checkout form. Pass
2.0	At the Place Order page disconnect the Internet. Reconnect and Click on 'Pay with card' button	User session should invalidated. user should get logged out with message ''your session has expired. Please Login again'' form. Pass

Table 4.21: **Test Cases for Req. ID FR21**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Click on the back button	User should be navigated back to the product listing page Pass

Table 4.22: **Test Cases for Req. ID FR22**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Leave the email address field blank	Error should be received and use should not be able to complete the payment process Pass
2.0	Enter an invalid string that does not conform to the regex pattern of an email address	Error should be received and use should not be able to complete the payment process Pass

Table 4.23: **Test Cases for Req. ID FR23**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Leave any of the required fields blank i.e Address Line 1, Pincode, Town/City, Country	Error should be received and use should not be able to complete the payment process Pass

Table 4.24: **Test Cases for Req. ID FR24**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Leave any of the required fields blanks i.e. Card Number Card Expiry Date Card CVV	Error should be received and use should not be able to complete the payment process Pass
2.0	Enter a previous date value in the card expiry date field	Error should be received and use should not be able to complete the payment process Pass

Table 4.25: **Test Cases for Req. ID FR25**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Click on the checkbox 'Remember Me'. Make Payment for the order. Relogin and initiate another order	Next time on the payment page, the saved card details should be presented to the user. Pass

Table 4.26: **Test Cases for Req. ID FR26**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Close the Stripe Payment form	User should be navigated back to the place Order page Pass

Table 4.27: **Test Cases for Req. ID FR27**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Place Order	Invoice cum payment receipt should be received and should not be empty Pass
2.0	Place Order. Disconnect network. Reconnect	Invoice cum payment receipt should not be received as user session will be invalid Pass

Table 4.28: **Test Cases for Req. ID FR28**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Place Order	Order invoice copy cum payment receipt should contain the Order ID Pass

Table 4.29: **Test Cases for Req. ID FR29**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Place Order	Order invoice copy cum payment receipt should contain product details such as the image, quantity of the product ordered, price per unit of the product and the total cost of the product Pass

Table 4.30: **Test Cases for Req. ID FR30**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Place Order	Order invoice copy cum payment receipt should contain total order quantity and total product value. Both of these values cannot be blank or empty Pass

Table 4.31: **Test Cases for Req. ID FR31**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Place Order using test card no 4000 0000 0000 0341 (test-mode)	transaction should fail and transaction error should be received Pass

Table 4.32: **Test Cases for Req. ID FR32**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Place order	Order confirmation notifications should be sent by email to the registered email address. If the registered email address and cardholder's email address are different, the order confirmation notification should be sent to the cardholder's email address as well. Pass

Table 4.33: **Test Cases for Req. ID FR33**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Login in as admin user.	Order data table should be presented. Order data should contain the order ID, total order value and the order timestamp. Pass
2.0	Click on any Order ID	Admin user should get to see all details related to the order i.e the email address, shipping address, products ordered, order quantity and value and order timestamp Pass

Table 4.34: **Test Cases for Req. ID FR34**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	On the admin main view click on the Manage Products link	Admin user should be navigated to the Manage Products page if the admin session is active Pass

Table 4.35: **Test Cases for Req. ID FR35**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Click on the Add Product button	Admin user should be navigated to the Add Product page if the admin user session is active Pass

Table 4.36: **Test Cases for Req. ID FR36**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Navigate to the manage products page	Product Catalog should be listed. It should contain details of the products i.e. the product image, name and price Pass

Table 4.37: **Test Cases for Req. ID FR37**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Click on Delete	Product should get removed form the view. User will have to relogin if already logged in. Pass

Table 4.38: **Test Cases for Req. ID FR38**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Click on update product	Admin user should be able to update product details. Refer test cases for Add Product Pass

Table 4.39: **Test Cases for Req. ID FR39**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Click on the link view order data	Admin user should be navigated to the main view Pass

Table 4.40: **Test Cases for Req. ID FR40**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Enter a blank value in any one of the fields product name, quantity or image	Error message should be received Pass
2.0	Enter a on-number value in any one of the product quantity field	Error message should be received Pass
3.0	Upload an image file size of more than 2 MB	Error message should be received Pass
4.0	Upload an image file with an invalid string for a filename	Error message should be received Pass

Table 4.41: **Test Cases for Req. ID FR41**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Select image and Upload	File image preview should be available. File name should be displayed On submit, filename should be stored in the database and the image to a file server Pass

Table 4.42: **Test Cases for Req. ID FR42**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Add a Product	Details should get saved to the database. A product ID should get auto-generated and stored in the database for referencing Pass

Table 4.43: **Test Cases for Req. ID FR43**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Click on logoff	Admin user should get logged off Pass

2.0	Click on Manage Products	Admin user should be navigated to the Manage Products page Pass
3.0	Click on View Orders	Admin user should be navigated to the View Orders Page Pass

Table 4.44: **Test Cases for Req. ID FR44**

Sr. No	**Test Cases**	**Expected Behavior & Status**
1.0	Enter blank values or invalid values in any of the fields	Error alerts should be received as described in the test cases above Pass

5 SYSTEM SECURITY MEASURES

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), both frequently referred to as "SSL", are cryptographic protocols that provide communications security over a Computer Network [6].

Go Code to run a secure HTTPS server using the "net/HTTP" package

```
    go http.ListenAndServe(":8090", http.RedirectHandler(↩
        "https://127.0.0.1:8091",301))
    err := http.ListenAndServeTLS(":8091", "tls/cert.pem"↩
        , "tls/key.pem", nil)
    log.Fatal(err)
}
```

TLS has been implemented in the project using the OpenSSL Client 1.0.1e

Command use to generate the TLS cert keys

openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes

Data contained in the cert key

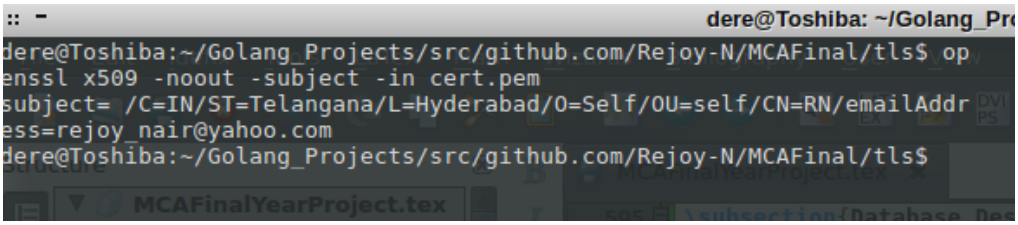


Figure 5.1: TLS Certificate Details - 1

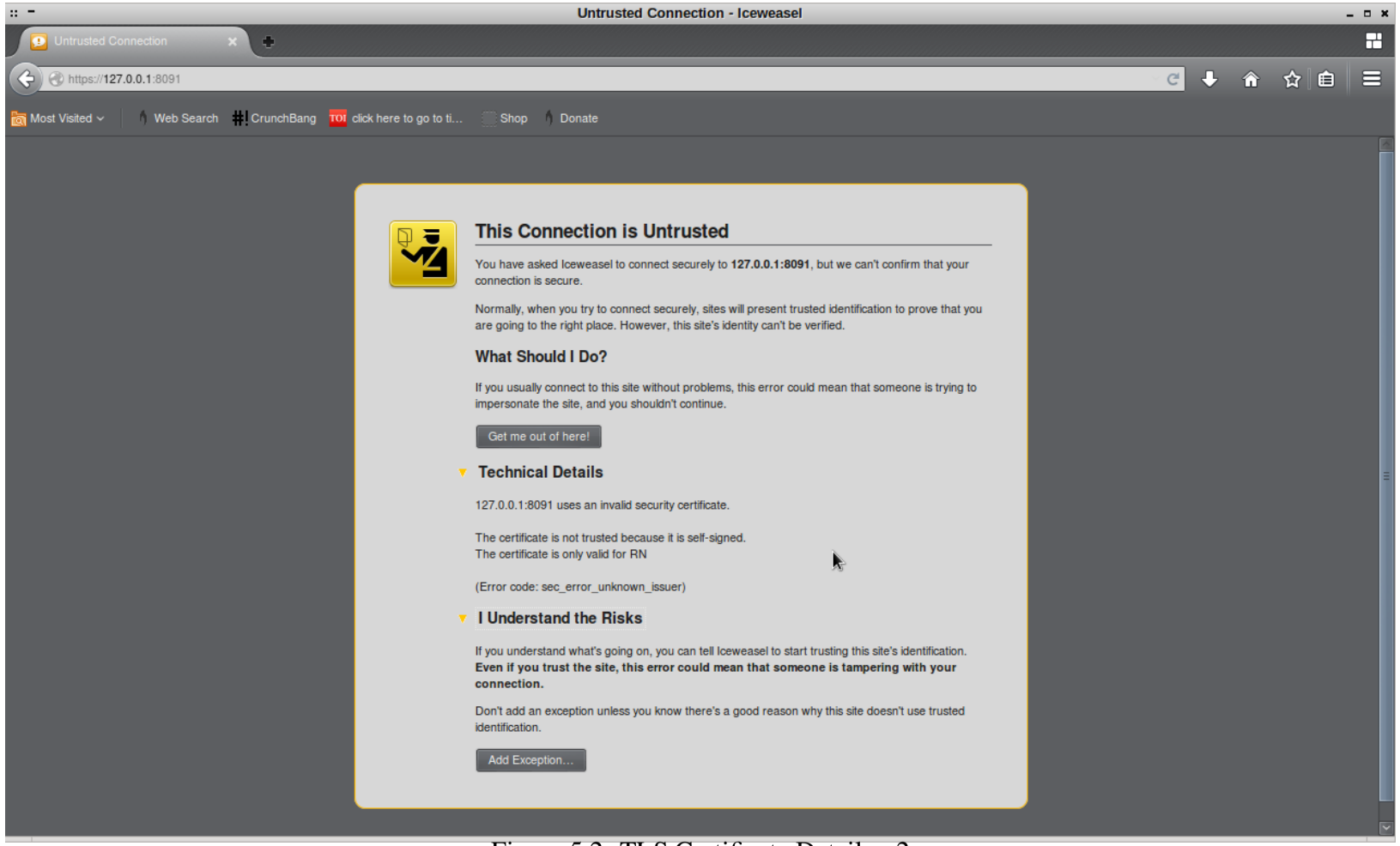


Figure 5.2: TLS Certificate Details - 2

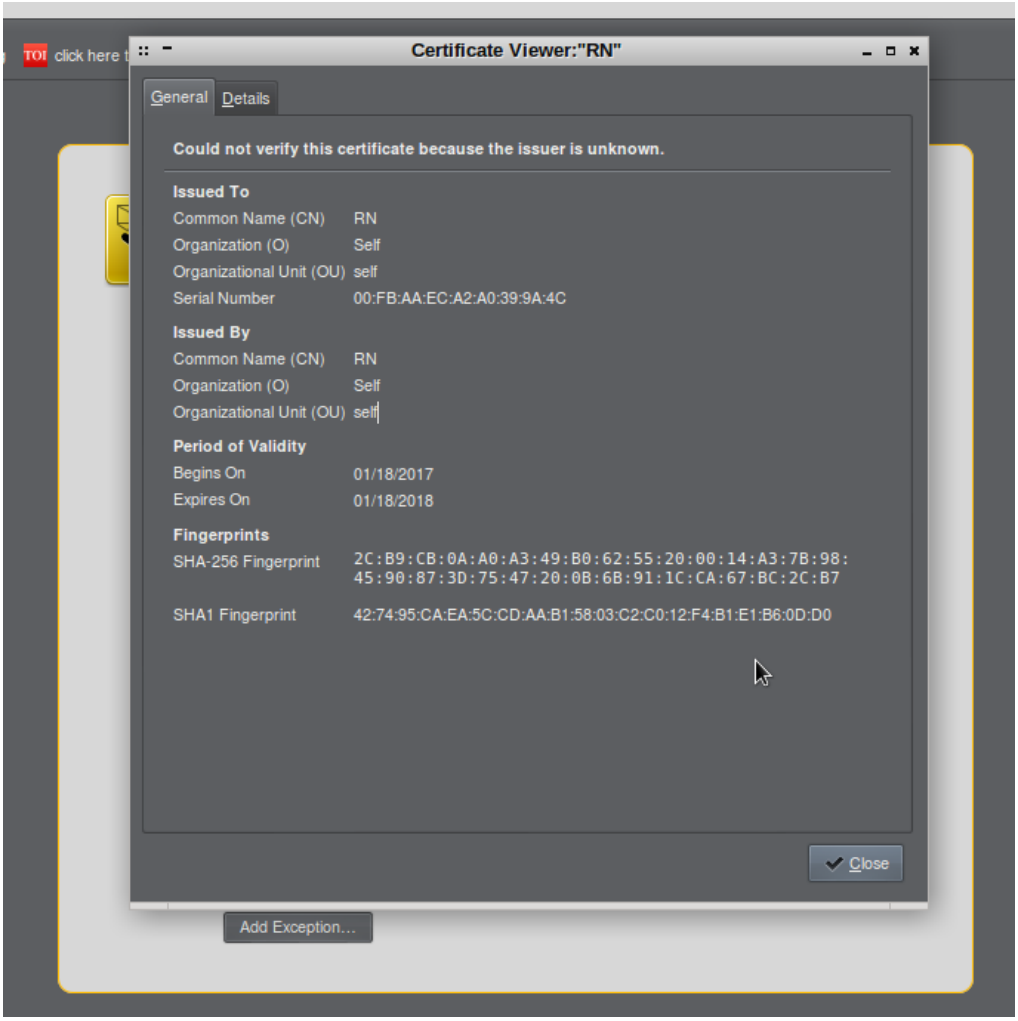


Figure 5.3: TLS Certificate Details - 3

The Transport Layer Security protocol aims primarily to provide privacy and data integrity between two communicating computer applications. When secured by TLS, connections between a client (e.g., a web browser) and a server have one or more of the following properties:

- The connection is private (or secure) because symmetric cryptography is used to encrypt the data transmitted. The keys for this symmetric encryption are generated uniquely for each connection and are based on a shared secret negotiated at the start of the session. The server and client negotiate the details of which encryption algorithm and cryptographic keys to use before the first byte of data is transmitted. The negotiation of a shared secret is both secure (the negotiated secret is unavailable to eavesdroppers and cannot be obtained, even by an attacker who places themselves in the middle of the connection) and reliable (no attacker can modify the communications during the negotiation without being detected).

- The identity of the communicating parties can be authenticated using public-key cryptography. This authentication can be made optional, but is generally required for at least one of the parties (typically the server).

- The connection ensures integrity because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission.

In addition to the properties above, careful configuration of TLS can provide additional privacy-related properties such as forward secrecy, ensuring that any future disclosure of encryption keys cannot be used to decrypt any TLS communications recorded in the past.

The TLS protocol comprises two layers: the TLS record protocol and the TLS handshake protocol.

Since applications can communicate either with or without TLS (or SSL), it is necessary for the client to indicate to the server the setup of a TLS connection. One of the main ways of achieving this is to use a different port number for TLS connections, for example port 443 for HTTPS.

Once the client and server have agreed to use TLS, they negotiate a stateful connection by using a handshaking procedure. The protocols use a handshake with an asymmetric cipher to establish cipher settings and a shared key for a session; the rest of the communication is encrypted using a symmetric cipher and the session key. During this handshake, the client and server agree on various parameters used to establish the connection‘s security:

- The handshake begins when a client connects to a TLS-enabled server requesting a secure connection and presents a list of supported cipher suites (ciphers and hash functions).
- From this list, the server picks a cipher and hash function that it also supports and notifies the client of the decision.
- The server usually then sends back its identification in the form of a digital certificate. The certificate contains the server name, the trusted Certificate Authority (CA) and the server‘s public encryption key.
- The client confirms the validity of the certificate before proceeding.
- To generate the session keys used for the secure connection, the client either:
    - encrypts a random number with the server‘s public key and sends the result to the server (which only the server should be able to decrypt with its private key); both parties then use the random number to generate a unique session key for subsequent encryption and decryption of data during the session
    - uses Diffie-Hellman key exchange to securely generate a random and unique session key for encryption and decryption that has the additional property of forward secrecy: if the server‘s private key is disclosed in future, it cannot be used to decrypt the current

session, even if the session is intercepted and recorded by a third party.

This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the session key until the connection closes. If any one of the above steps fail, the TLS handshake fails, and the connection is not created.

Digital Certificates
A digital certificate certifies the ownership of a public key by the named subject of the certificate, and indicates certain expected usages of that key. This allows others (relying parties) to rely upon signatures or on assertions made by the private key that corresponds to the certified public key.

TLS typically relies on a set of trusted third-party certificate authorities to establish the authenticity of certificates. Trust is usually anchored in a list of certificates distributed with user agent software and can be modified by the relying party. As a consequence of choosing X.509 certificates, certificate authorities and a public key infrastructure are necessary to verify the relation between a certificate and its owner, as well as to generate, sign, and administer the validity of certificates.

6 COST ESTIMATION OF THE PROJECT

Estimate Product Size and Resources
The correlation of program size with development time is only moderately good for engineering teams and organizations. However, for individual engineers, the correlation is generally quite high. Therefore, the PSP starts with engineers estimating the sizes of the products they will personally develop. Then, based on their personal size and productivity data, the engineers estimate the time required to do the work. In the PSP, these size and resource estimates are made with the PROBE method [7].

Size Estimating with PROBE PROBE stands for PROxy Based Estimating and it uses proxies or objects as the basis for estimating the likely size of a product. With PROBE, engineers first determine the objects required to build the product described by the conceptual design. Then they determine the likely type and number of methods for each object. They refer to historical data on the sizes of similar objects they have previously developed and use linear regression to determine the likely overall size of the finished product. The example object size data in figure below show the five size ranges the PSP uses for objects. Since object size is a function of programming style, the PROBE method shows engineers how to use the data on the programs they have personally developed to generate size ranges for their personal use. Once they have estimated the sizes of the objects, they used linear regression to estimate the total amount of code they plan to develop. To use linear regression, the engineers must have historical data on estimated versus actual program size for at least three prior programs.

C++ Object Sizes in LOC per Method					
Category	**Very Small**	**Small**	**Medium**	**Large**	**Very Large**
Calculation	2.34	5.13	11.25	24.66	54.04
Data	2.60	4.79	8.84	16.31	30.09
I/O	9.01	12.06	16.15	21.62	28.93
Logic	7.55	10.98	15.98	23.25	33.83
Set-up	3.88	5.04	6.56	8.53	11.09
Text	3.75	8.00	17.07	36.41	77.66

Figure 6.1: C++ Object Size Data

Any reference material for Object Size Data for Go Programs was not found. Hence the C++ object size data was used in the estimation.

Go Files	Function	LOC	Avg LOC per Function
main	15	627	41.80
data	7	121	17.29
cookie	5	71	14.20
product	8	161	20.13
order	4	135	33.75
email	1	34	34.00

Figure 6.2: Effort details - Go Programs

HTML/JS Files	LOC
index	25
main	22
signup	50
internal	233
placeorder	101
admin	49
vieworder	108
manageproduct	76
addproduct	164
deleteproduct	105
showproduct	112
shoppingCart.Js	119

Figure 6.3: Effort details - HTML/JS Programs

Size Measures : Lines of Code (LOC) Since the time it takes to develop a product is largely determined by the size of that product, when using the PSP, engineers first estimate the sizes of the products they plan to develop. Then, when they are done, they measure the sizes of the products they produced. This provides the engineers with the size data they need to make accurate size estimates. However, for these data to be useful, the size measure must correlate with the development time for the product. While lines of code (LOC) is the principal PSP size measure, any size measure can be used that provides a reasonable correlation between development time and product size. It should also permit automated measurement of actual product size.

The PSP uses the term “logical LOC“ to refer to a logical construct of the programming language being used. Since there are many ways to define logical LOC, engineers must precisely define how they intend to measure LOC. In this project LOC is simply the actual number of lines of codes as provided by the IDE.

7 PROJECT CHART, GANTT CHART

There cannot be a concept of a The program (or project) evaluation and review technique (PERT) chart for software development work carried out by a single developer as there is no other option but to carry out the work sequentially. In that sense, the PERT chart and the Gantt chart would be the same. PSP however has methods are introduced in a series of seven process versions. These versions are labeled PSP0 through PSP3, and each version has a similar set of logs, forms, scripts, and standards, as shown in Figure. The process scripts define the steps for each part of the process, the logs and forms provide templates for recording and storing data, and the standards guide the engineers as they do the work.

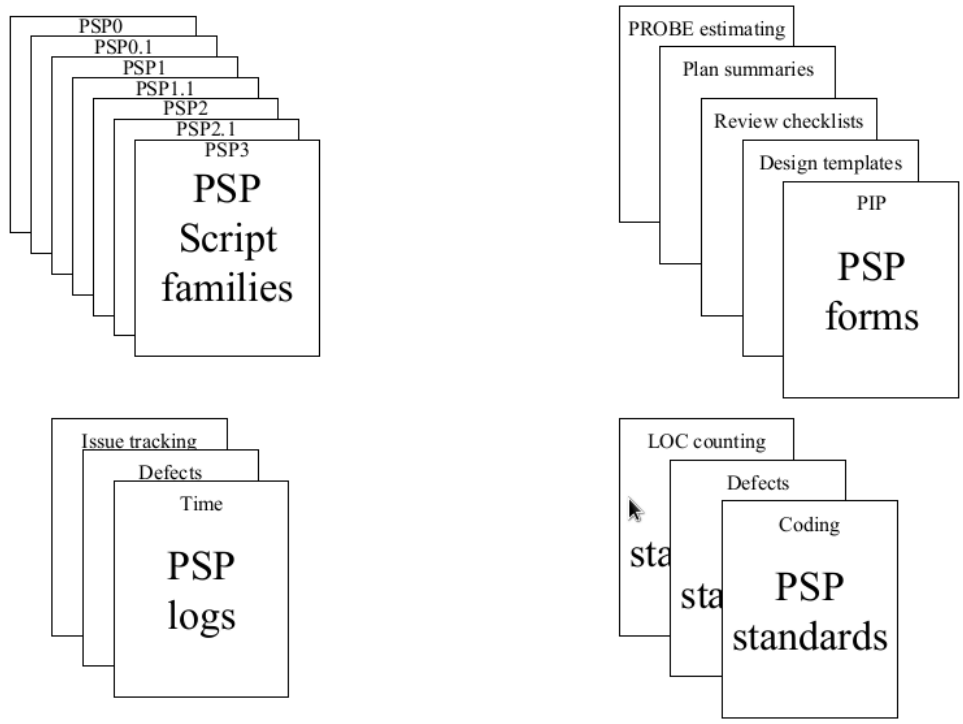


Figure 7.1: PSP Process Elements

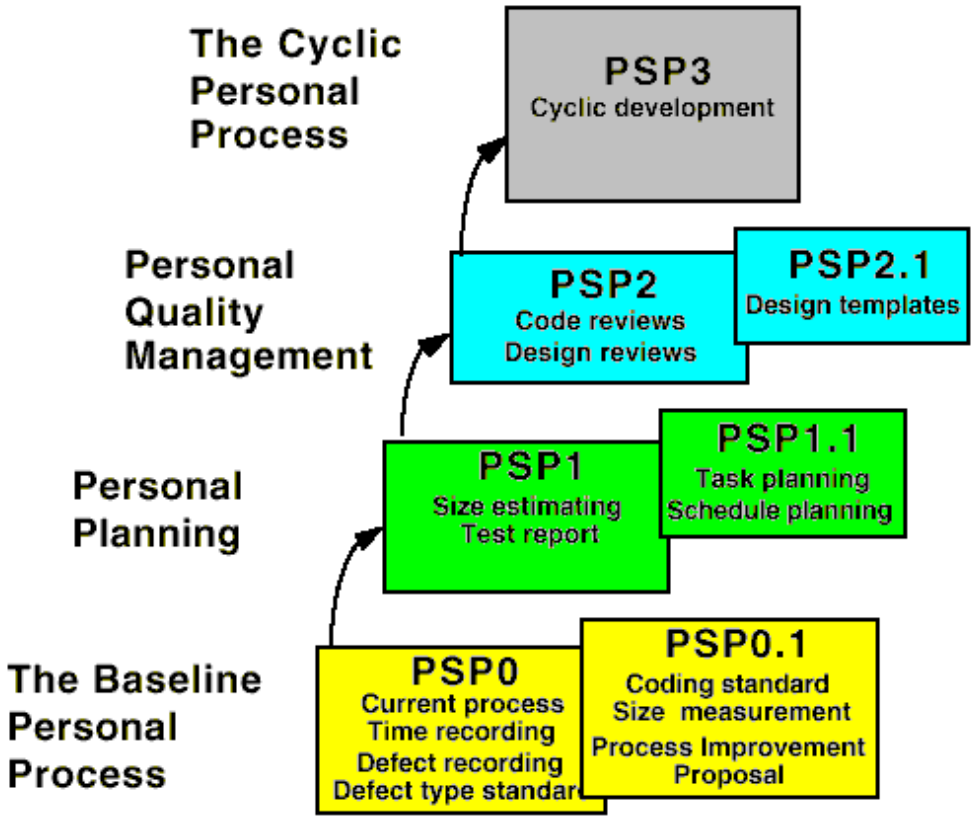
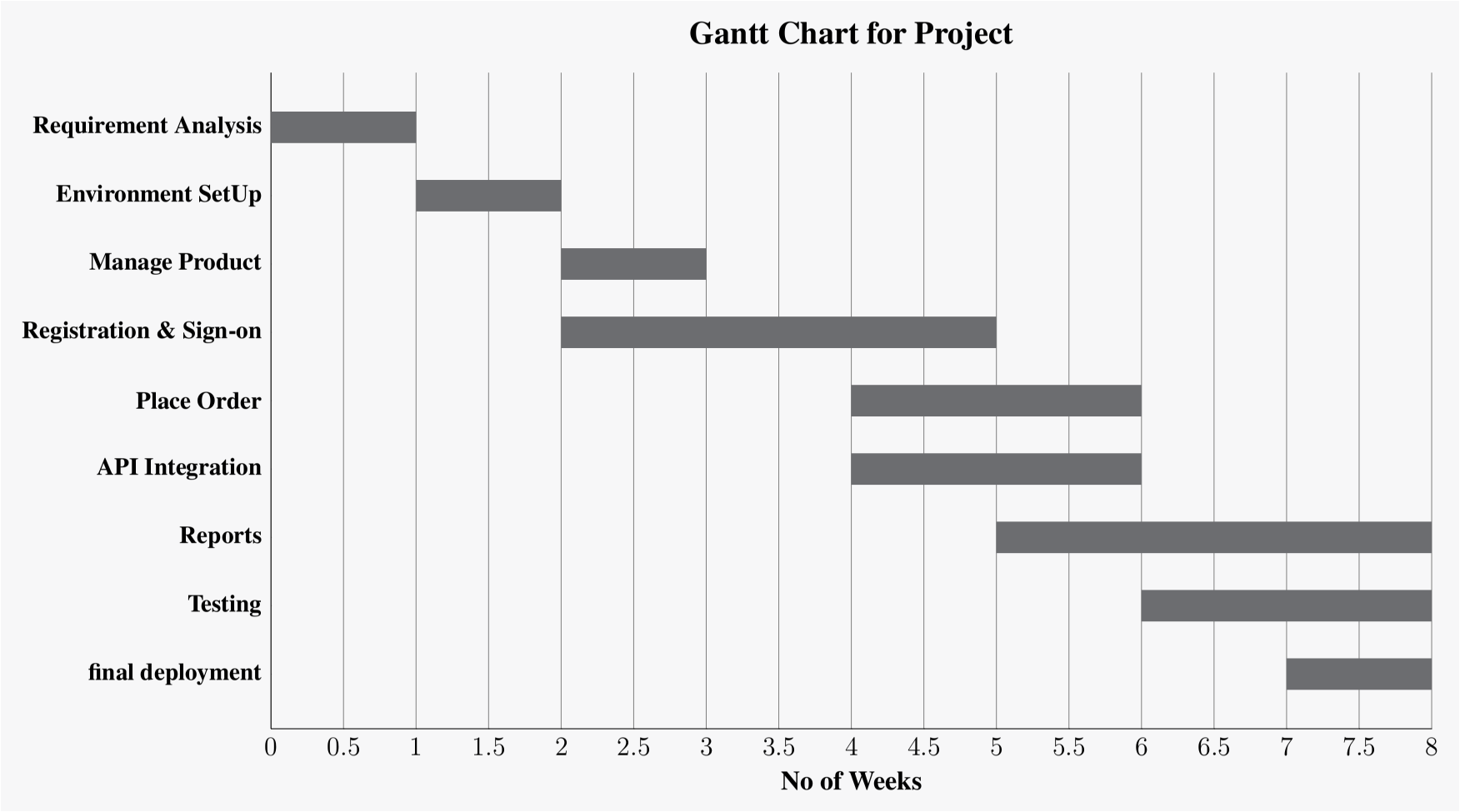


Figure 7.2: PSP Process Elements in the PSP Process


Gantt Chart for Project
Requirement Analysis
Environment SetUp
Manage Product
Registration & Sign-on
Place Order
API Integration
Reports
Testing
final deployment
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6
6.5
7
7.5
8
No of Weeks

8 FUTURE SCOPE AND FURTHER ENHANCEMENT

The existing website can be developed further by adding more functionalities viz.,

- Forgot Password functionality that allows users to reset password
- Single-Sign On
- Use of apt Keywords in Meta-tags that allow fo better results in Google Search
- Filter Products
- Persistent Cart
- Cancel Order
- Print PDF
- Add more Product attributes
- Different Modes of Payment Like Digital Wallets
- Responsive Web Design (RWD)
- Location services based on Wifi or GPS
- Rate & Review Product
- Chat interface

The above is only an indicative list. Most of the features for retail website can be implemented using Go.

An important aspect of this project is that the entire development has been done on a i686 m/c equipped with just 512 MB RAM. This is an important factor in increasing functionality and scalability of the web app.

9 BIBLIOGRAPHY

[1] https://en.wikipedia.org/wiki/Online_shopping

[2] http://www.indiaretailing.com/2016/08/30/retail/online-shopping-trends-facts-figures-on-indian-e-comm-sector

[3] https://en.wikipedia.org/wiki/Personal_software_process

[4] https://talks.golang.org/2012/splash.article

[5] https://en.wikipedia.org/wiki/User_interface_design

[6] https://en.wikipedia.org/wiki/Transport_Layer_Security

[7] Pittsburgh, PA 15213-3890. The Personal. Software. Process. SM. (PSP. SM.) CMU/SEI-2000-TR-022. ESC-TR-2000-022. Watts S. Humphrey. November 2000.

10 APPENDICES

List of Figures

List of Tables

11 GLOSSARY

Glossary

4G 4th Generation (4G) is a mobile communications standard intended to replace 3G, allowing wireless Internet access at a much higher speed.. 4

Algorithm a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.. 71

API In computer programming, an application programming interface (API) is a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.. 7, 8, 17, 34, 50, 52

asymmetric cipher *See* public-key cryptography. 71

Authentication the process or action of verifying the identity of a user or process.. 12

B2C Business-to-consumer (B2C) is an Internet and electronic commerce (e-commerce) model that denotes a financial transaction or online sale between a business and consumer.. 1

Broadband a high-capacity transmission technique using a wide range of frequencies, which enables a large number of messages to be communicated simultaneously.. 4

CA Certificate Authorities (CA) issue Digital Certificates. A certificate authority (CA) is a trusted entity that issues electronic documents that verify a digital entity‘s identity on the Internet. The electronic documents, which are called digital certificates, are an essential part of secure communication and play an important part in the public key infrastructure (PKI).. 71

Client (in a network) a desktop computer or workstation that is capable of obtaining information and applications from a server.. 71

Closures In computer science, a closure is a function that has an environment of its own. Inside this environment, there is at least one bound variable.. 28

CMM The Capability Maturity Model (CMM) is a methodology used to develop and refine an organization’s software development process. The model describes a five-level evolutionary path of increasingly organized and systematically more mature processes.. 25

Compiler a program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer. 28

Composition In computer science, function composition is an act or mechanism to combine simple functions to build more complicated ones. Object composition is a way to combine simple objects or data types into more complex ones.. 30

Computer Network a set of computers connected together for the purpose of sharing resources.. 1, 68

Concurrency In computer science, concurrency is the execution of several instruction sequences at the same time. In an operating system, this happens when there are several process threads running in parallel. These threads may communicate with each other through either shared memory or message passing.. 30

Correlation A correlation is a single number that describes the degree of relationship between two variables. 73

Cryptographic protocols A security protocol (cryptographic protocol or encryption protocol) is an abstract or concrete protocol that performs a security-related function and applies cryptographic methods, often as sequences of cryptographic primitives.. 68

Decryption Decryption is the process of taking encoded or encrypted text or other data and converting it back into text that you or the computer can read and understand. This term could be used to describe a method of un-encrypting the data manually or with un-encrypting the data using the proper codes or keys.. 72

Diffie-Hellman key exchange Diffie–Hellman key exchange (D–H) is a specific method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as originally conceptualized by Ralph Merkle and named after Whitfield Diffie and Martin Hellman.. 72

Digital Certificate In cryptography, a public key certificate (also known as a digital certificate or identity certificate) is an electronic document used to prove the ownership of a public key.. 71

e-commerce *See* Electronic Commerce (also e-commerce). 4

EDI Electronic Data Interchange (EDI) is the electronic interchange of business information using a standardized format; a process which allows one company to send information to another company electronically rather than with paper.. 1

Electronic Commerce (also **e-commerce)** commercial transactions conducted electronically on the Internet.. 1

Email messages distributed by electronic means from one computer user to one or more recipients via a network.. 1, 34, 40

Encryption the process of converting information or data into a code, especially to prevent unauthorized access.. 71

Error handling Error handling refers to the anticipation, detection, and resolution of programming, application, and communications errors.. 30

First-class functions Specifically, this means the language supports passing functions as arguments to other functions, returning them as the values from other functions, and assigning them to variables or storing them in data structures. It means that functions are objects, with a type and a behaviour. They can be dynamically built, passed around as any other object, and the fact that they can be called is part of their interface.. 28

Forward Secrecy In cryptography, forward secrecy (FS; also known as perfect forward secrecy) is a property of secure communication protocols in which compromise of long-term keys does not compromise past session keys. Forward secrecy protects past sessions against future compromises of secret keys or passwords.. 71, 72

Garbage collection the automatic process of making space in a computer's memory by removing data that is no longer required or in use.. 27

Go-live (of a computer system) become operational.. 4

Handshake The handshaking process usually takes place in order to establish rules for communication when a computer sets about communicating with a foreign device. When a computer communicates with another device like a modem, printer, or network server, it needs to handshake with it to establish a connection.. 71

Hash functions A hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes.. 71

Hosting store (a website or other data) on a server or other computer so that it can be accessed over the Internet.. 8

HTTPS HTTPS (HTTP over SSL or HTTP Secure) is the use of Secure Socket Layer (SSL) or Transport Layer Security (TLS) as a sublayer under regular HTTP application layering. HTTPS encrypts and decrypts user page requests as well as the pages that are returned by the Web server.. 12, 71

i686 i686 is widely used to describe 32-bit P6 processor architecture which is compatible with Pentium Pro/II and has it's instruction set.. 5

Identifier An identifier is the user-defined name of a program element. It can be a namespace, class, method, variable or interface. Identifiers are symbols used to uniquely identify a program element in the code. They are also used to refer to types, constants, macros and parameters.. 28

Inheritance In object-oriented programming, inheritance enables new objects to take on the properties of existing objects.. 30

Internet a global computer network providing a variety of information and communication facilities, consisting of interconnected networks using standardized communication protocols.. 1

Internet Marketing Internet marketing, or online marketing, refers to advertising and marketing efforts that use the Web and email to drive direct sales via electronic commerce, in addition to sales leads from Web sites or emails.. 1

Inventory Management Inventory management is the supervision of non-capitalized assets (inventory) and stock items. A component of supply chain management, inventory management supervises the flow of goods from manufacturers to warehouses and from these facilities to point of sale.. 1

Linear regression In statistics, linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X.. 73

Linker a program used with a compiler or assembler to provide links to the libraries needed for an executable program.. 28

Linux an open-source operating system modelled on UNIX.. 5

LOC Source lines of code (SLOC), also known as lines of code (LOC), is a software metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code.. 73

Marketplace the arena of commercial dealings. An online marketplace (or online e-commerce marketplace) is a type of e-commerce site where product or service information is provided by multiple third parties, whereas transactions are processed by the marketplace operator.. 2

Message Authentication Code In cryptography, a message authentication code (MAC) is a short piece of information used to authenticate a message—in other words, to confirm that the message came from the stated sender (its authenticity) and has not been changed in transit (its integrity).. 71

Mobile Commerce M-commerce (mobile commerce) is the buying and selling of goods and services through wireless handheld devices such as cellular telephone and personal digital assistants (PDAs).. 1

Namespaces A namespace in computer science (sometimes also called a name scope), is an abstract container or environment created to hold a logical grouping of unique identifiers or symbols.. 28

Network security Network security is protection of the access to files and directories in a computer network against hacking, misuse and unauthorized changes to the system.. 3

Object file An object file is a file containing object code, meaning relocatable format machine code that is usually not directly executable.. 28

Online Transaction Processing Online transaction processing, or OLTP, is a class of information systems that facilitate and manage transaction-oriented applications, typically for data entry and retrieval transaction processing.. 1

OOP Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure.. 3

Open source denoting software for which the original source code is made freely available and may be redistributed and modified.. 27

OpenSSL OpenSSL is a software library to be used in applications that need to secure communications over computer networks against eavesdropping or need to ascertain the identity of the party at the other end.. 68

Operating System the low-level software that supports a computer‘s basic functions, such as scheduling tasks and controlling peripherals.. 7

Payment Gateway A payment gateway is a merchant service provided by an e-commerce application service provider that authorizes credit card or direct payments processing for e-businesses, online retailers, bricks and clicks, or traditional brick and mortar.. 7, 8, 17, 34, 50, 52

Paypal Paypal is an electronic commerce (e-commerce) company that facilitates payments between parties through online funds transfers.. 1

Persistent Persistent shopping carts save a customer‘s cart contents across sessions through “persistent cookies.“ A cookie is a small text file stored on a user‘s computer. In computer science, persistence refers to the characteristic of state that outlives the process that created it. This is achieved in practice by storing the state as data in computer data storage. Programs have to transfer data to and from storage devices and have to provide mappings from the native programming-language data structures to the storage device data structures. 13

Private Key A private key is a tiny bit of code that is paired with a public key to set off algorithms for text encryption and decryption. It is created as part of public key cryptography during asymmetric-key encryption and used to decrypt and transform a message to a readable format.. 72

PROBE Proxy-Based Estimating (PROBE) is an estimating process used in the Personal Software Process (PSP) to estimate size and effort.. 73

PSP The Personal Software Process (PSP) is a structured software development process that is intended (planned) to help software engineers better understand and improve their performance by tracking their predicted and actual development of code.. 8, 25, 73

RDBMS Relational Database management System (RDBMS) is a database management system based on relational model defined by E.F.Codd. Data is stored in the form of rows and columns. The relations among tables are also stored in the form of the table.. 3

Real-time transaction processing In a real-time processing system, transactions are processed immediately as they occur without any delay to accumulate transactions.. 1

Reflection In computer science, reflection is the ability of a computer program to examine, introspect, and modify its own structure and behavior at runtime.. 29

Search Engine a program that searches for and identifies items in a database that correspond to keywords or characters specified by the user, used especially for finding particular sites on the World Wide Web.. 1

Session the period of activity between a user logging in and logging out of a (multi-user) system.. 12

Session key A session key is an encryption and decryption key that is randomly generated to ensure the security of a communications session between a user and another computer or between two computers. Session keys are sometimes called symmetric keys, because the same key is used for both encryption and decryption.. 71

Social Media websites and applications that enable users to create and share content or to participate in social networking.. 4

Social Networks a dedicated website or other application which enables users to communicate with each other by posting information, comments, messages, images, etc.. 1

Software Requirement Specifications store (A software requirements specification (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide.. 10

SSL Secure Sockets Layer (SSL) is the standard security technology for establishing an encrypted link between a web server and a browser.. 1

Stateful Connection A stateful connection is one in which some information about a connection between two systems is retained for future use. 71

Statically typed Enforcement of type rules at compile time rather than at run time. Static typing catches more errors at compile time than dynamic typing.. 27

Supply Chain Management Supply chain management (SCM) is the oversight of materials, information, and finances as they move in a process from supplier to manufacturer to wholesaler to retailer to consumer.. 1

Symbol table In computer science, a symbol table is a data structure used by a language translator such as a compiler or interpreter, where each identifier (a.k.a. symbol) in a program‘s source code is associated with information relating to its declaration or appearance in the source.. 28

Symmetric cipher Symmetric ciphers are the oldest and most used cryptographic ciphers. In a symmetric cipher, the key that deciphers the ciphertext is the same as (or can be easily derived from) the key enciphers the clear text. This key is often referred to as the secret key.. 71

TLS Transport layer security (TLS) is a protocol that provides communication security between client/server applications that communicate with each other over the Internet.. 3, 68

TLS handshake protocol The Transport Layer Security (TLS) Handshake Protocol is responsible for the authentication and key exchange necessary to establish or resume secure sessions.. 71

TLS record protocol The Transport Layer Security (TLS) Record protocol secures application data using the keys created during the Handshake.. 71

TSP n combination with the personal software process (PSP), the team software process (TSP) provides a defined operational process framework that is designed to help teams of managers and engineers organize projects and produce software products that range in size from small projects of several thousand lines of code (KLOC) to very large projects greater than half a million lines of code. 25

Type information In computer programming, run-time type information refers to a mechanism that exposes information about an object‘s data type at runtime. Run-time type information can apply to simple data types, such as integers and characters, or to generic types.. 28

Type-safe Type-Safe is code that accesses only the memory locations it is authorized to access, and only in well-defined, allowable ways.. 28

URL A URL (Uniform Resource Locator), as the name suggests, provides a way to locate a resource on the web, the hypertext system that operates over the internet.. 28

UUID A UUID (Universal Unique Identifier) is a 128-bit number used to uniquely identify some object or entity on the Internet.. 24

Variadic functions In computer science, an operator or function is variadic if it can take a varying number of arguments; that is, if its arity is not fixed.. 28

Web Browser a computer program with a graphical user interface for displaying HTML files, used to navigate the World Wide Web.. 1

Web design Web design is the process of creating websites. It encompasses several different aspects, including webpage layout, content production, and graphic design. While the terms web design and web development are often used interchangeably, web design is technically a subset of the broader category of web development.. 3

Web security Web application security is a branch of Information Security that deals specifically with security of websites, web applications and web services. At a high level, Web application security draws on the principles of application security but applies them specifically to Internet and Web systems.. 3

Web server A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients.. 8

World Wide Web an information system on the Internet which allows documents to be connected to other documents by hypertext links, enabling the user to search for information by moving from one document to another.. 1

X.509 certificates In cryptography, X.509 is an important standard for a public key infrastructure (PKI) to manage digital certificates and public-key encryption and a key part of the Transport Layer Security protocol used to secure web and email communication.. 72