PROJECT REPORT

ON

AutoMat

To be developed to fulfil the requirements for

Major Project (CA133)

Submitted to

Dr. Jaiteg Singh

(Dean)

Dr. Jaswinder Singh

(Associate Professor)

Department of Computer Application

Chitkara University, Punjab

Under the supervision of

Dr. Jaswinder Singh
(Associate Professor)

Submitted by

Akul Sikand	Dev Gupta	Deevanshi
2110992509	2110992538	2110992536

CERTIFICATE

This is to certify that the report on software project titled "AutoMat" submitted by the student team comprising Akul Sikand (2110992509), Dev Gupta (2110992538), Deevanshi (2110992536) to the Department of Computer Applications, Chitkara University, Punjab in partial fulfilment for the completion of the course Major Project (CA133) in the fifth semester of Bachelor of Computer Applications is a Bona fide record of work carried out by the team under my supervision.

Date:

DR. JASWINDER SINGH
Associate Professor
Department of Computer Application
Chitkara University, Rajpura, Punjab

DR. JAITEG SINGH
Professor and Dean, Department of Computer Applications
Chitkara University, Punjab

TABLE OF CONTENTS

Abstract of The Project

The "Automat" project is a comprehensive vending machine management system designed to streamline the user experience and enhance administrative control. The system comprises two main components: the administrative side and the user side. The primary goal is to embed an intuitive interface on vending machines and efficiently manage inventory through an admin dashboard. The technologies employed include HTML, CSS, PHP, Laravel, and MySQL, ensuring a robust and dynamic web application.

Administrative Side:
The admin interface begins with a secure login page leading to a dashboard with controls for users, machines, and promotions. The 'Users' section enables the admin to view and manage user information, including email and user type. Admins can add or remove users as needed. In the 'Machines' section, admins can monitor machine details and perform actions such as adding or removing machines. The 'Promotions' section facilitates the management of advertisements.

User Side:
On the user side, the vending machine interface presents users with advertisements upon opening. A product listing displays various items with names and prices, each featuring an 'Add to Cart' option for selection. The top-right corner houses a 'Cart' button, allowing users to view selected items, adjust quantities, and navigate back to shopping. The backend simultaneously records user selections.

Technological Stack:
The project utilizes HTML and CSS for front-end development, PHP and Laravel for server-side logic, and MySQL for database management. This technology stack ensures a seamless and responsive user interface, while Laravel facilitates efficient backend processing.

Key Considerations:
Security measures have been implemented to safeguard user data and prevent common web vulnerabilities. The user interface is designed to be intuitive and responsive, promoting a positive user experience. The database schema is structured for optimal performance, with proper indexing and relationships. The admin dashboard includes data visualization tools for comprehensive information presentation.

Conclusion:
"Automat" addresses the complexities of vending machine management, offering a user-friendly interface for consumers and a powerful admin dashboard for efficient control. By leveraging a robust technological stack and adhering to secure coding practices, the project aims to set new standards in vending machine management systems.

Profile of problems assigned:

The Profile of Problems Assigned in my project encapsulates a multifaceted set of challenges to be addressed. It encompasses ensuring secure and efficient user identification and authentication, differentiation of user types with appropriate permissions, efficient monitoring and management of vending machine inventory, dynamic addition or removal of machines, smooth integration and display of advertisements, secure transaction processing, optimization of ad loading performance on the vending machine interface, and the implementation of an intuitive and responsive user interface for product selection and cart management. Additionally, the project tackles real-time backend data updates, logging admin actions for auditing purposes, and the seamless integration of HTML, CSS, PHP, Laravel, and MySQL technologies to create a cohesive and efficient system. Usability and user experience challenges include designing interfaces that are user-friendly, intuitive for both admins and users, and responsive across various devices for a consistent user experience. This comprehensive profile outlines the intricacies and scope of the project, providing a roadmap for addressing these challenges.

Study of existing system

It involves a thorough analysis of current vending machine management systems. This examination encompasses the overview of existing systems, focusing on their features and limitations, user interface evaluations with an emphasis on design and usability, scrutiny of inventory management practices including restocking methods, examination of current advertisement integration strategies, investigation into security measures protecting user data and transactions, analysis of user and machine management procedures, exploration of the technological stack commonly employed, and a review of user feedback and satisfaction. By comprehensively understanding the strengths and weaknesses of existing systems, the project aims to leverage successful practices while addressing and improving upon identified shortcomings, thus informing the development of a more efficient and user-friendly vending machine management system.

System Requirements

a. Product Definition

a) Problem Statement:
The existing vending machine management systems lack a cohesive and user-friendly solution for both administrators and users. Challenges include inefficient user and machine management, suboptimal inventory control, and limitations in the integration and display of advertisements. Additionally, security concerns surrounding user data and transactions need robust addressing. The current systems lack a comprehensive and intuitive approach to product selection and cart management for users interacting with vending machines.

b) Function to be Provided:
The "Automat" project aims to provide a holistic vending machine management system. The functions include streamlined user and machine management, enhanced inventory control, efficient promotion handling, and a responsive vending machine interface. Users can easily browse products, add items to a cart, and perform secure transactions. Admins can manage users, machines, and promotions seamlessly through a user-friendly dashboard.

c) Processing Environment: H/W, S/W.
The system requires standard hardware components including databases for data storage, and vending machines with compatible interfaces that we are not showing now but it's are idea to be implemented in future. The software components consist of HTML and CSS for the front-end, PHP and Laravel for server-side logic, and MySQL for database management. This technological stack ensures a robust and scalable processing environment.

d) Solution Strategy:
The solution strategy involves the integration of HTML, CSS, PHP, Laravel, and MySQL to create a cohesive and efficient vending machine management system. User interfaces are designed to be intuitive and responsive, addressing existing challenges in user and machine management. Security measures are implemented to protect user data and transactions. The promotion handling system is developed to seamlessly integrate and display advertisements.

e) Acceptance Criteria:
The project's success is contingent upon meeting several key criteria. These include the successful implementation of user and machine management functions, efficient inventory control, seamless promotion handling, and the creation of a user-friendly vending machine interface. Security measures must be robust and prevent unauthorized access. The system's performance should meet predefined standards, ensuring a positive user experience for both administrators and end-users.

This product definition outlines the core aspects of the "Automat" project, providing a clear understanding of the problems addressed and the solutions proposed.

b. Feasibility Analysis

The feasibility analysis for the "Automat" project involves a thorough examination of technical, operational, and economic factors to ensure the viability and success of the system.

Technical Feasibility:
The technical feasibility of the project is evident in the successful integration of HTML, CSS, PHP, Laravel, and MySQL. These technologies are widely supported and offer a robust framework for building dynamic web applications. Compatibility checks have been conducted to ensure smooth interactions between the chosen technologies, and the proposed solution aligns with industry best practices.

Operational Feasibility:
Operationally, the system is designed to streamline vending machine management for administrators and enhance the shopping experience for users. The user interfaces are intuitive, reducing the learning curve for both admins and end-users. The operational benefits include efficient inventory control, responsive interfaces, and a user-friendly admin dashboard for seamless management.

Economic Feasibility:
From an economic standpoint, the "Automat" project is feasible as it leverages open-source technologies like PHP and Laravel, reducing licensing costs. The benefits derived from improved vending machine management, increased user satisfaction, and potential revenue from advertisements contribute to the economic viability of the project.

c. Project Plan

a) Team Structure:
The project will be executed by a multidisciplinary team comprising developers, designers, and a project manager. Responsibilities include front-end and back-end development, database management, UI/UX design, and project coordination. The team structure ensures a collaborative approach to address the diverse aspects of the project. UI/UX is managed by Deevanshi, Database by Akul, and Establishment of connection is managed by Dev and the problems faced by each member are managed collectively.

b) Development Schedule:
The development schedule is outlined in phases to ensure a systematic approach. The initial phase involves requirement gathering and planning. Subsequent phases include front-end and back-end development, database integration, testing, and deployment. A detailed timeline ensures that each phase is completed efficiently, with regular check-ins and milestones to track progress.

c) Programming Languages and Development Tools:
The project utilizes a combination of HTML and CSS for the front-end, providing a visually appealing and responsive user interface. PHP and Laravel are employed for server-side logic,

ensuring efficient processing and data management. MySQL serves as the database management system, storing and retrieving data seamlessly. Development tools such as code editors i.e. visual studio code, version control systems are employed to maintain code quality and facilitate collaboration among team members.

The feasibility analysis confirms the technical, operational, and economic viability of the "Automat" project, while the project plan outlines a structured approach to development, ensuring efficient utilization of resources and timely delivery.

System Requirement Specifications

a. Developing / Operating / Maintenance Environments

The "Automat" system is designed to operate, develop, and undergo maintenance in specific environments.

Developing Environment:
The development environment requires standard development tools such as code editors (e.g., Visual Studio Code, Sublime Text), version control systems (e.g., Git), and testing frameworks (e.g., PHP Unit for PHP). Developers should have access to a development server running PHP, Laravel, and MySQL for local testing and debugging.

Operating Environment:
The operating environment necessitates a web server compatible with PHP and Laravel o host the application. Users will access the system through modern web browsers like Chrome, Firefox, or Safari. The vending machines should have interfaces compatible with HTML and CSS for seamless interaction.

Maintenance Environment:
Maintenance tasks involve regular updates, bug fixes, and potential system enhancements. Maintenance requires access to the production server, version control for code management, and a backup mechanism for data protection. The maintenance team should be familiar with PHP, Laravel, and MySQL.

b. External Interface and Data Flows

a) User Display and Report Format, User Command Summary:
The user interface will be displayed on vending machines and administrator dashboards. For users, the display includes advertisements, product listings with names and prices, and a shopping cart. Admins access a dashboard with controls for user and machine management. The command summary involves user actions like selecting products, adjusting quantities in the cart, and admins managing users, machines, and promotions.

b) High-Level DFD and Data Dictionary:
A high-level Data Flow Diagram (DFD) illustrates the flow of data within the system. It encompasses user interactions, inventory updates, and communication between the admin dashboard and vending machines. The Data Dictionary provides detailed definitions of data entities, including user profiles, machine details, product information, and transaction records.

c) Functional and Performance Specifications:

Functional Specifications:

- User Management: Admins can add, delete, and view user details.

- Machine Management: Admins can add, delete, and view machine details.
- Promotion Management: Admins can manage advertisements and promotions.
- Vending Machine Interface: Users can view ads, select products, and manage a shopping cart.
- Cart Management: Users can adjust product quantities and complete transactions.

Performance Specifications:

- Ad Loading: Ads on the vending machine interface should load efficiently to avoid delays.
- Transaction Processing: Transactions should be secure and processed in real-time.
- System Responsiveness: The system should respond promptly to user interactions on both admin and user interfaces.
- Database Performance: The database should be optimized for efficient data retrieval and storage.

These specifications outline the necessary environments for development, operation, and maintenance, as well as the external interfaces and data flows within the "Automat" system. The functional and performance specifications provide a clear roadmap for system functionality and performance expectations.

Profile of problem Assigned statement

Name of Student	Technology Used	Role Assigned
Akul	HTML, CSS, PHP, JavaScript	Backend and Database connection
Dev Gupta		Cart Management, Routing
Deevanshi		Front-end and Login Page, Promotions Tab(database)

Table 1: Profile of role assigned

1. Akul:

- Technology Used: HTML, CSS, PHP, JavaScript

- Role Assigned: Backend and Database Connection

- Explanation: Akul is responsible for the backend development of the project, which involves server-side logic and database connectivity. HTML and CSS are used for structuring and styling the web pages, while PHP is employed for server-side scripting. JavaScript is likely used for client-side scripting and enhancing user interactions. Additionally, Akul may be involved in setting up and managing the database, ensuring smooth communication between the frontend and backend systems.

2. Dev Gupta:

- Technology Used: HTML, CSS, PHP, JavaScript

- Role Assigned: Cart Management, Routing, Testing

- Explanation: Dev Gupta has a multifaceted role. Firstly, he is responsible for managing the shopping cart functionality, which includes handling user interactions related to adding, removing, and updating items in the cart. Secondly, Dev Gupta is in charge of defining and managing the routing within the application, ensuring proper navigation. Lastly, he is involved in testing, which implies verifying that the implemented features work correctly and meet the project requirements.

3. Deevanshi:

- Technology Used: HTML, CSS, PHP, JavaScript

- Role Assigned: Front-end and Login Page, Promotions Tab, Testing

- Explanation: Deevanshi's primary responsibility is the front-end development of the project. This involves creating visually appealing and user-friendly interfaces using HTML, CSS, and JavaScript. Deevanshi is specifically tasked with designing the login page and managing the Promotions Tab, likely involving the retrieval and display of promotional content from the database. In addition to her frontend roles, Deevanshi is also involved in testing, ensuring that the frontend features meet quality standards and work as intended.

Design

1. Detailed DFD: -

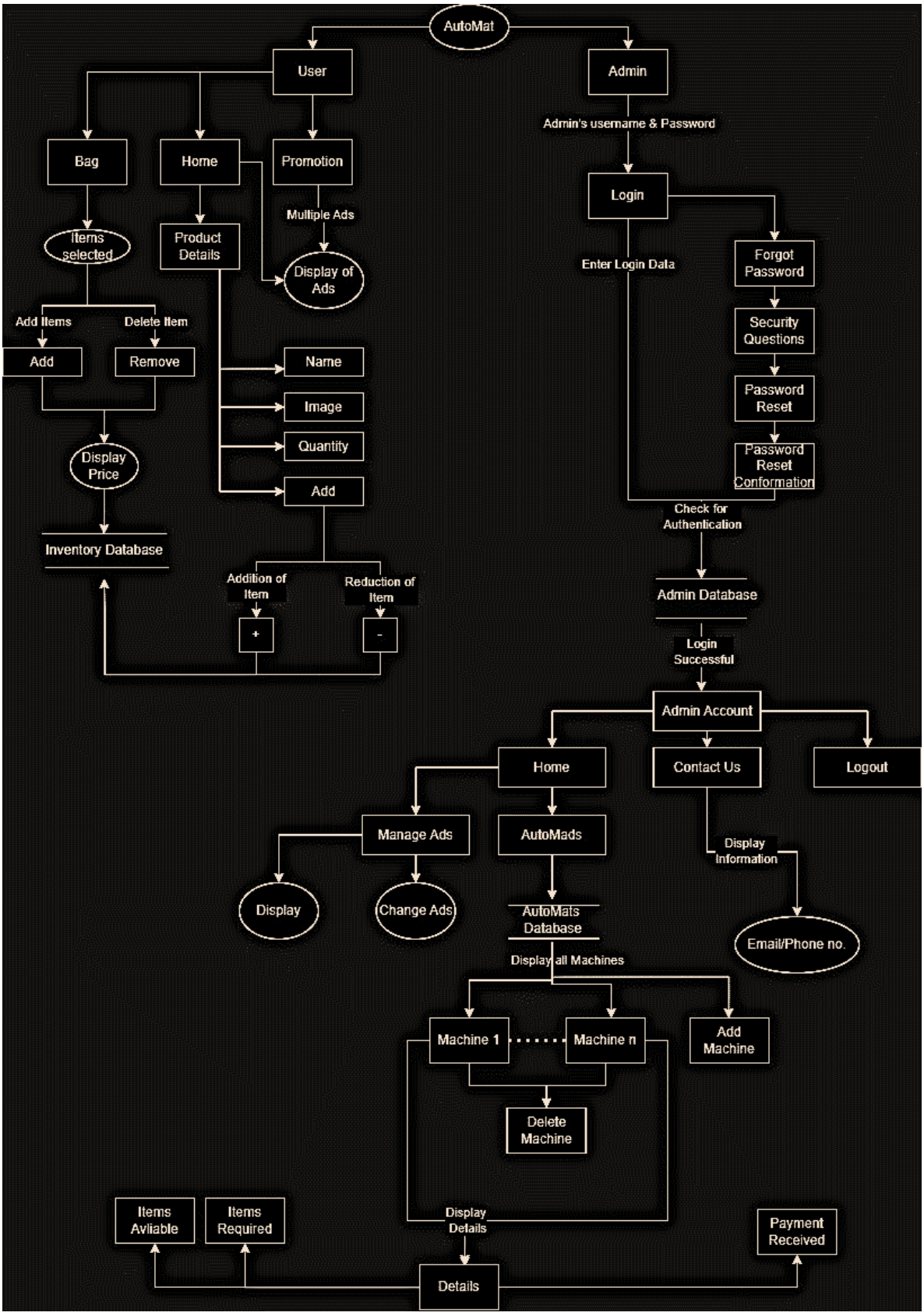

Table 2: Detailed Data Flow Diagram

Name of our project is "Automat". In the project as shown in the DFD:
On the admin side, first of all, there opens a login page through which admin can get over to its dashboard. Various controls are available on the admin's dashboard for users, machines and promotions.
Through the 'users', admin is able to see the information(email, user type) about the users which includes admin and various sellers and from here admin can add and delete an user.
Through the 'machines', admin is able to see the information about the machines available and from here admin can add and delete a machine.
Through the 'promotions', ads can be managed.

On the admin side, when a machine is opened, first of all it will show up ads on the top. Then, there is listing of various products available on which product's name is written and its price is mentioned. There is an option of add to cart on each product through which you can select product you want to buy. On the top right corner, there is a button of cart which shows all the selected items. From there also, you can increase or decrease your product's quantity and also, you can come back to your shopping.

Meanwhile, at the backend, your chosen products are shown.

2. External Interface Design

- **User Interface Design**
-

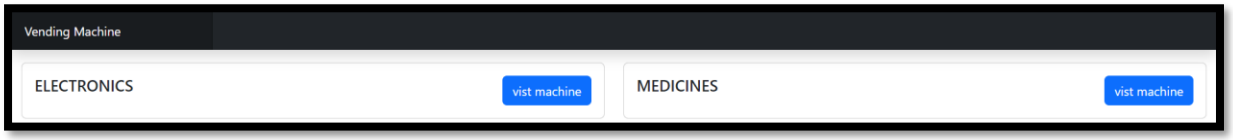

Figure 1: List of machines

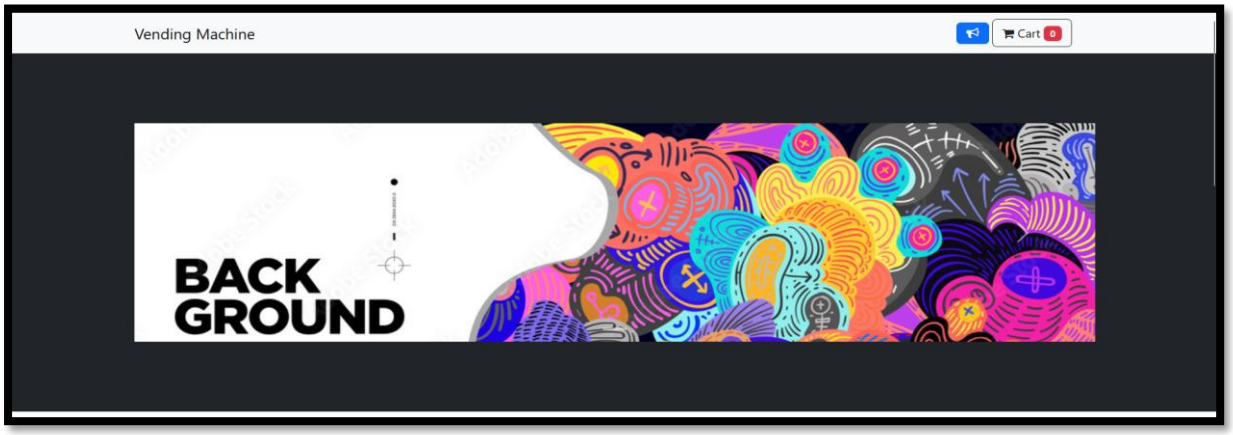

Figure 2: Promotions Section on "Electronics" Machine

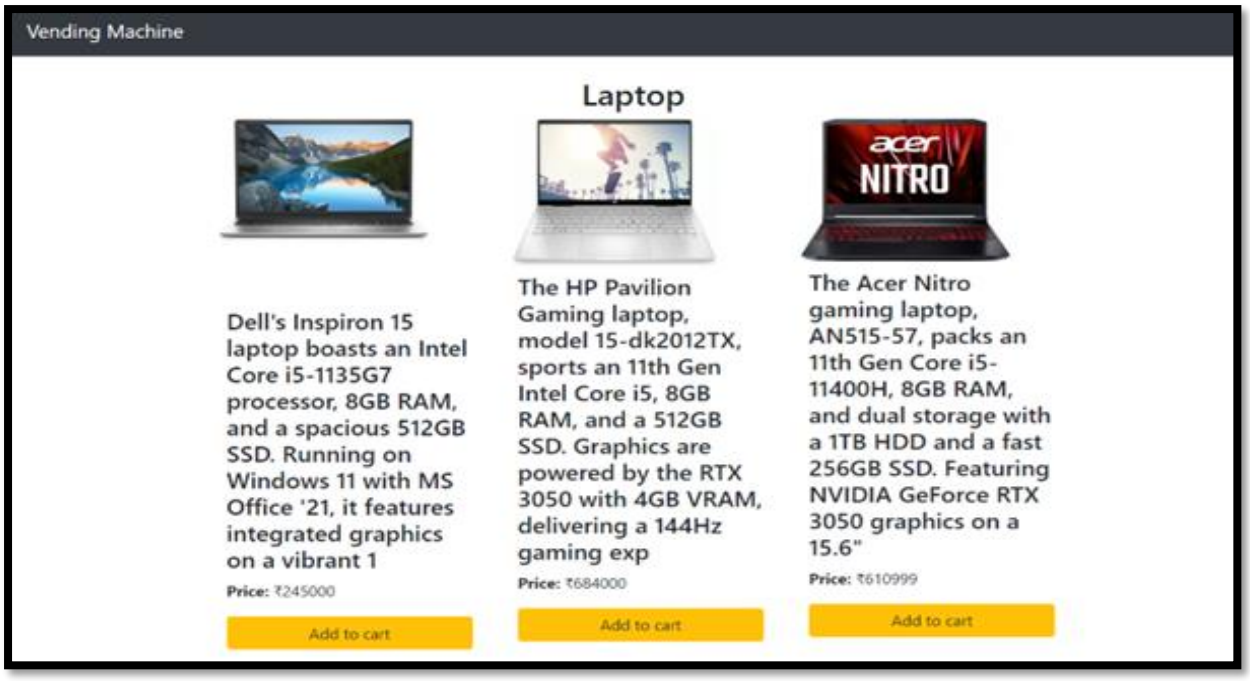

Figure 3: Items available in "Electronics" Machine

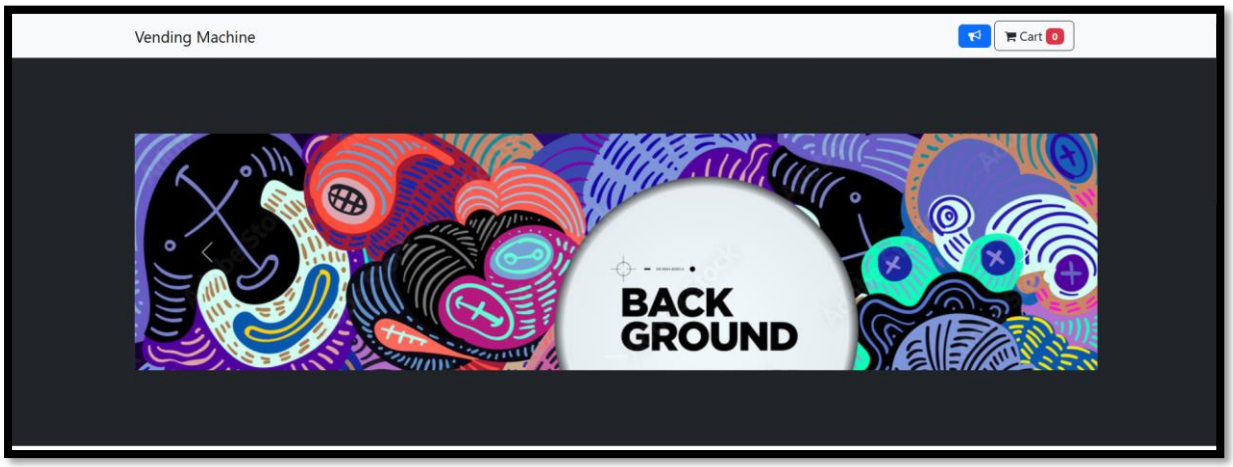

Figure 4: Promotions Section on "Medicines" Machine

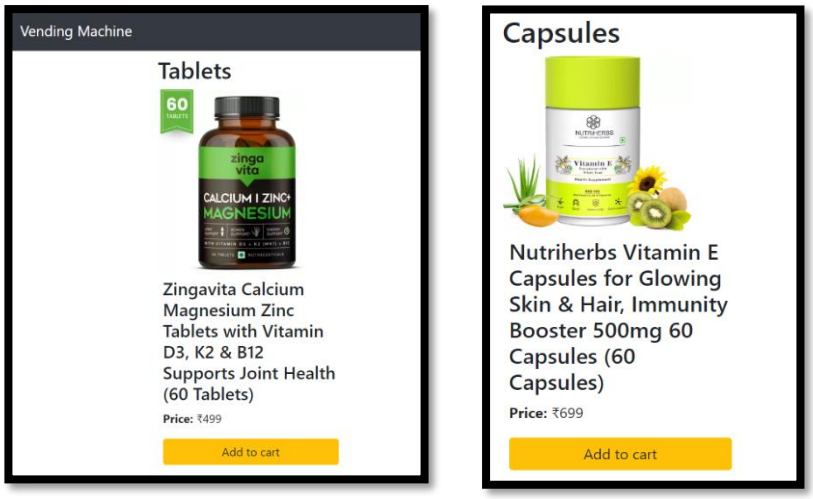

Figure 5: Items available on "Medicines" Machine

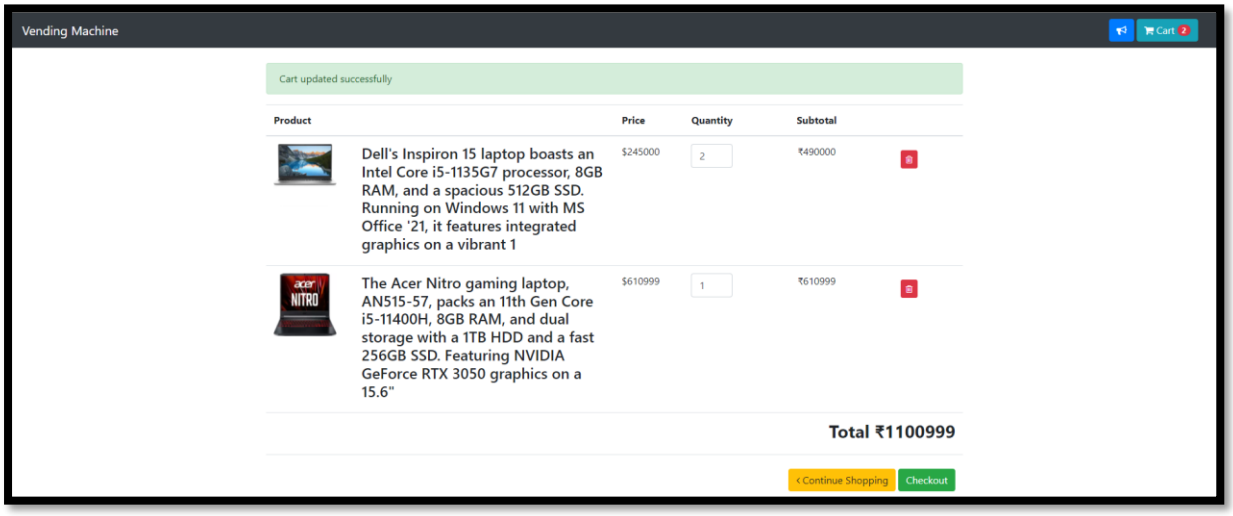

Figure 6: Items in cart (full view)

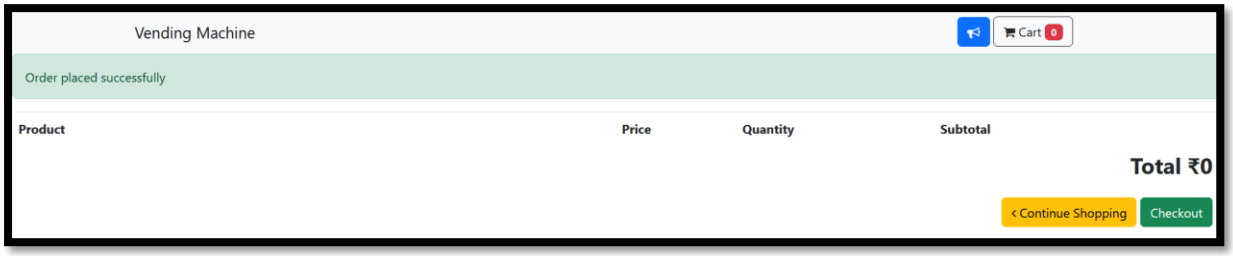

Figure 7: After successfully placing order

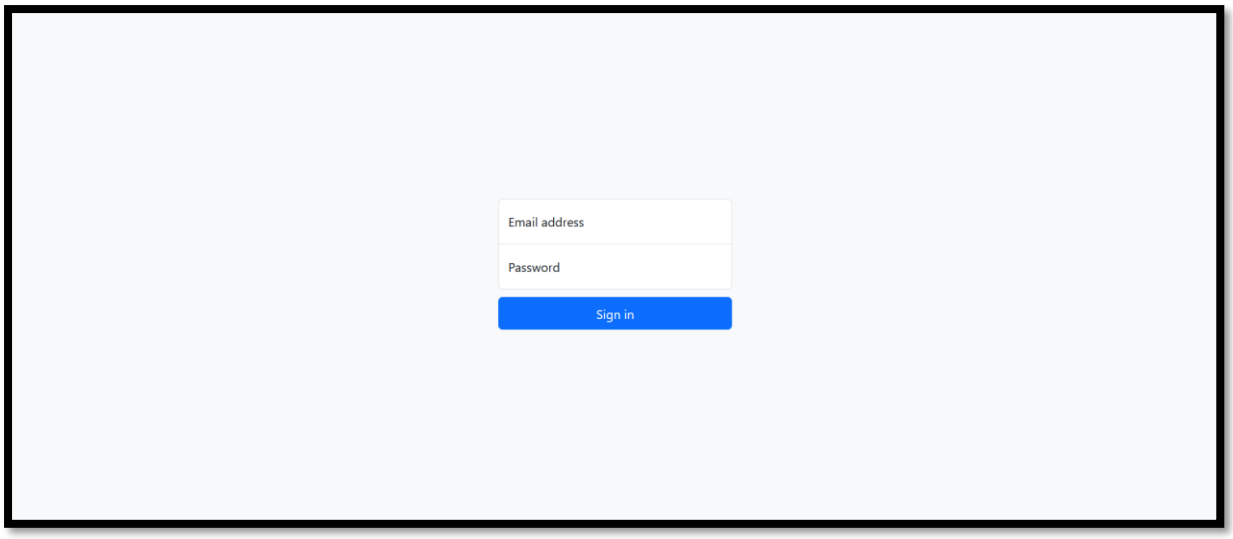

Figure 8: Login Page (Admin/Seller)

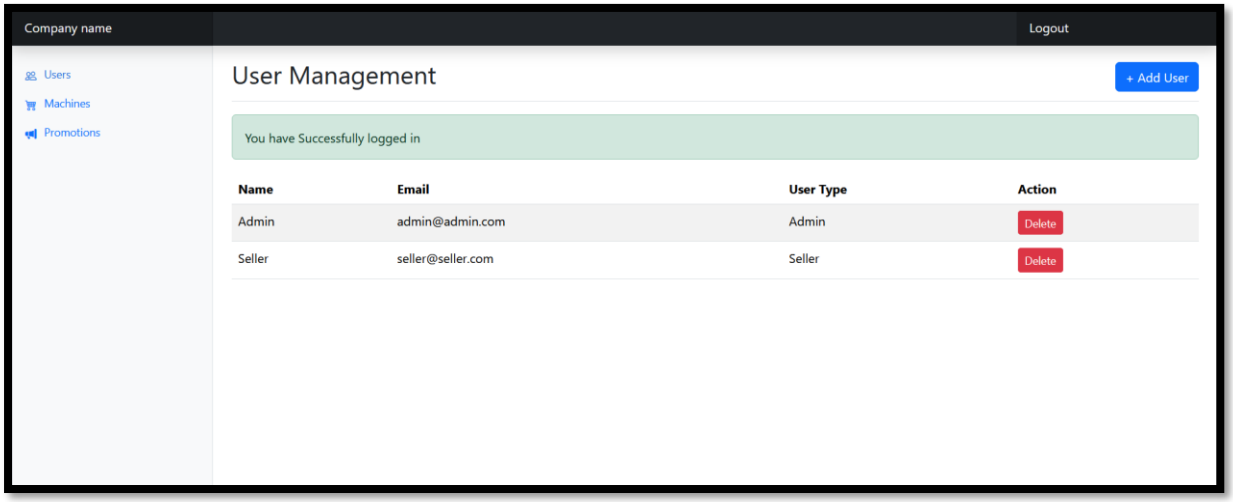

Figure 9: Admin Dashboard

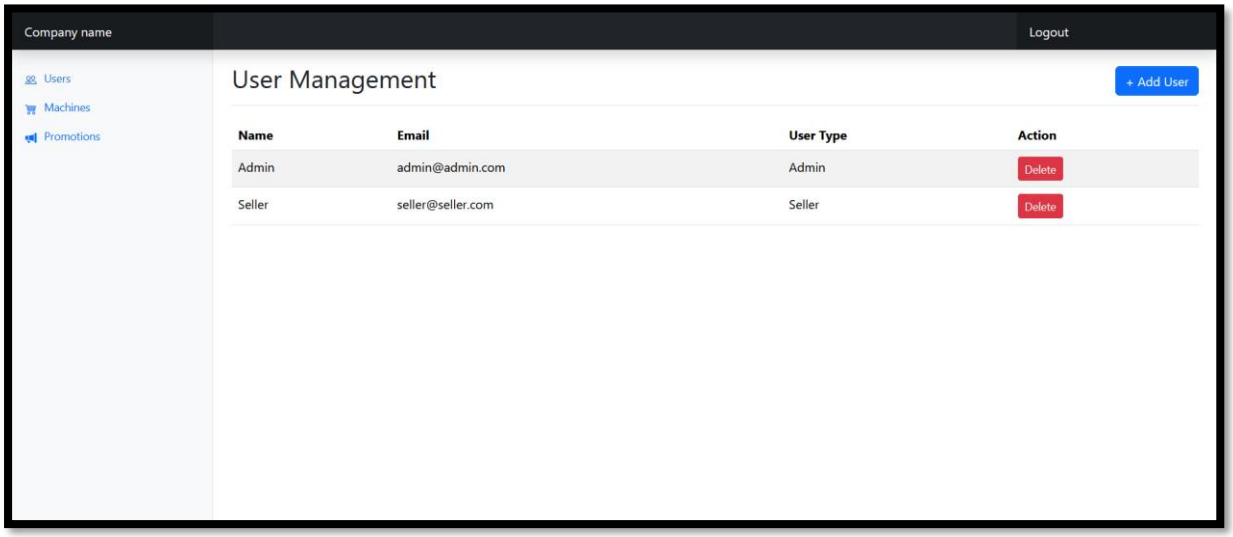

Figure 10: Number of users (Admin/sellers)

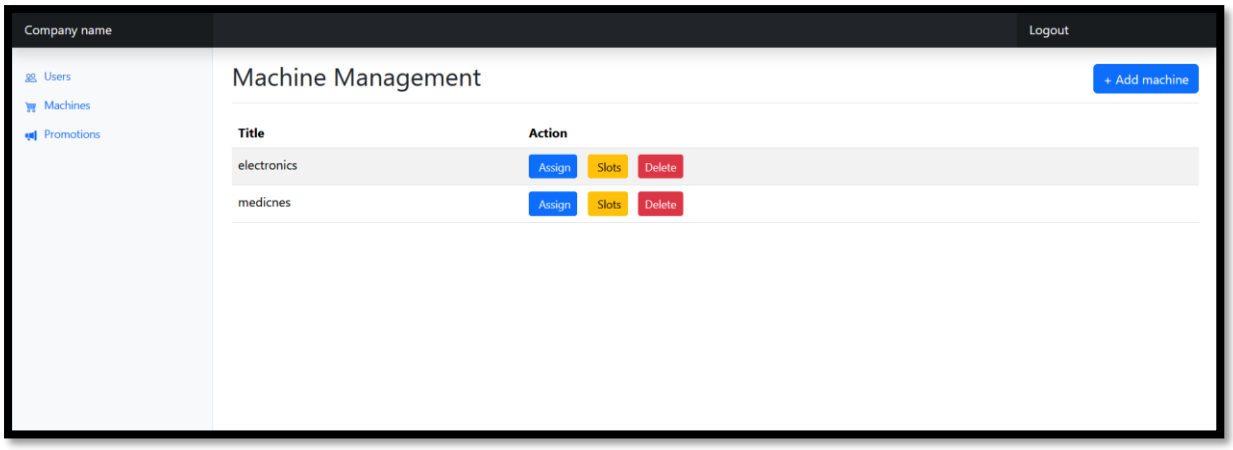

Figure 11: Number of Machines Listed

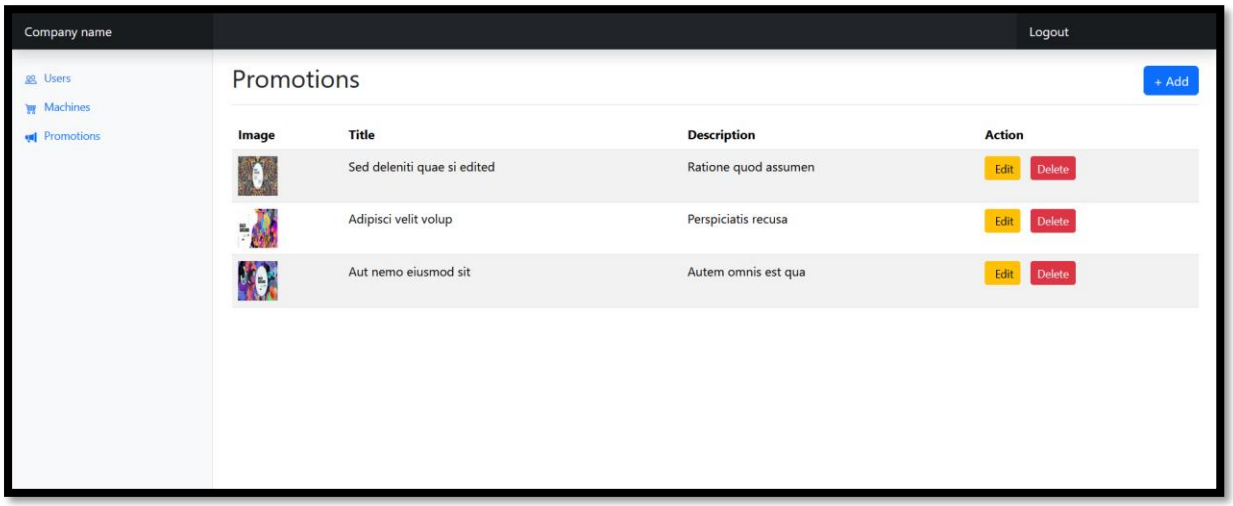

Figure 12: Number of on-going promotions

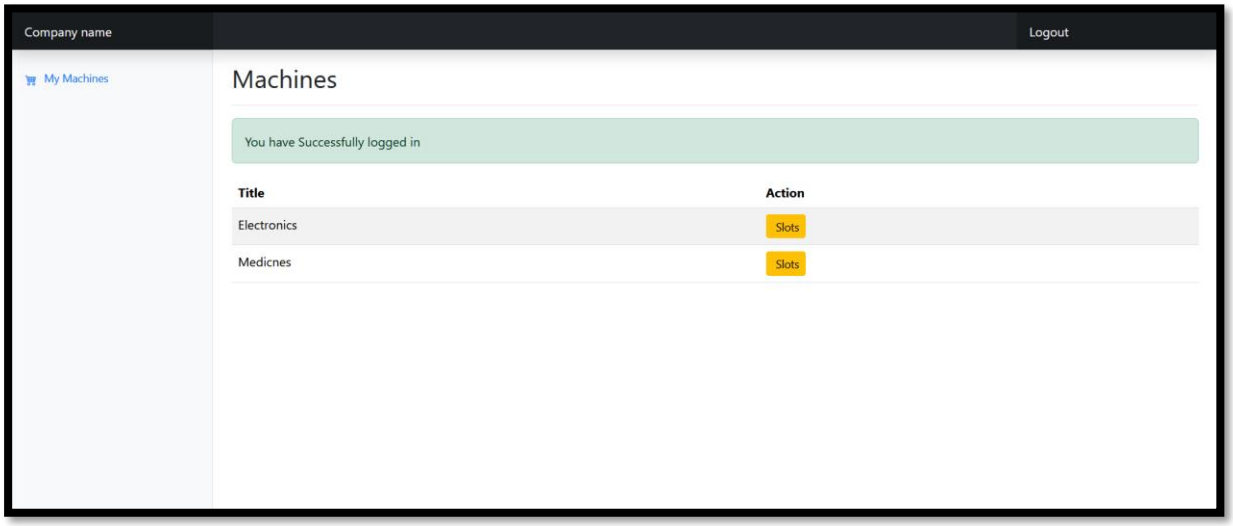

Figure 13: Seller's Dashboard

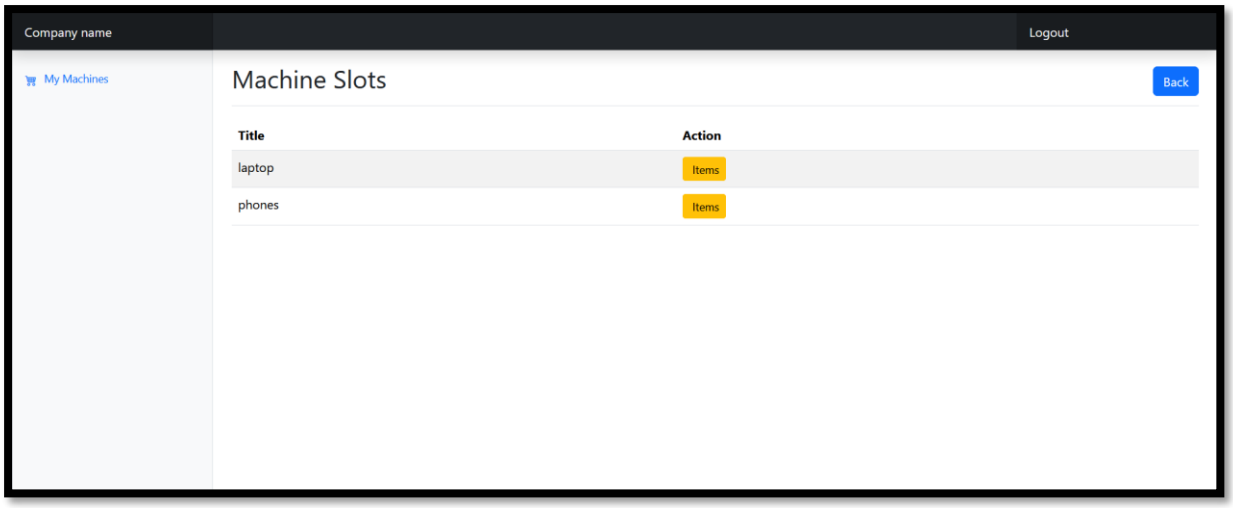

Figure 14: Items in Machine Slots

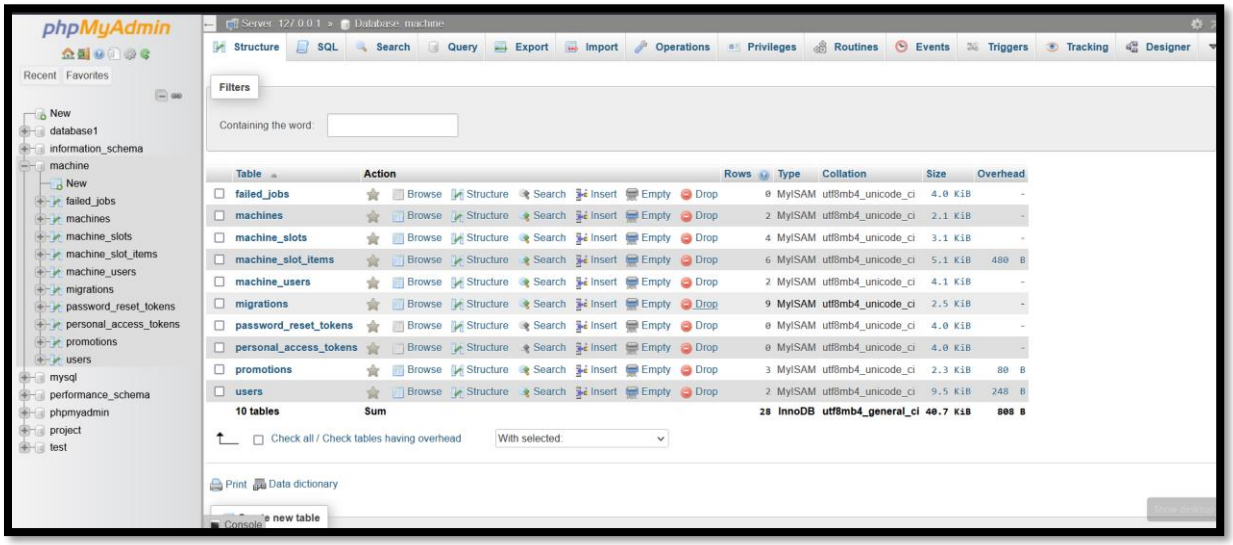

Figure 15: Database Overview

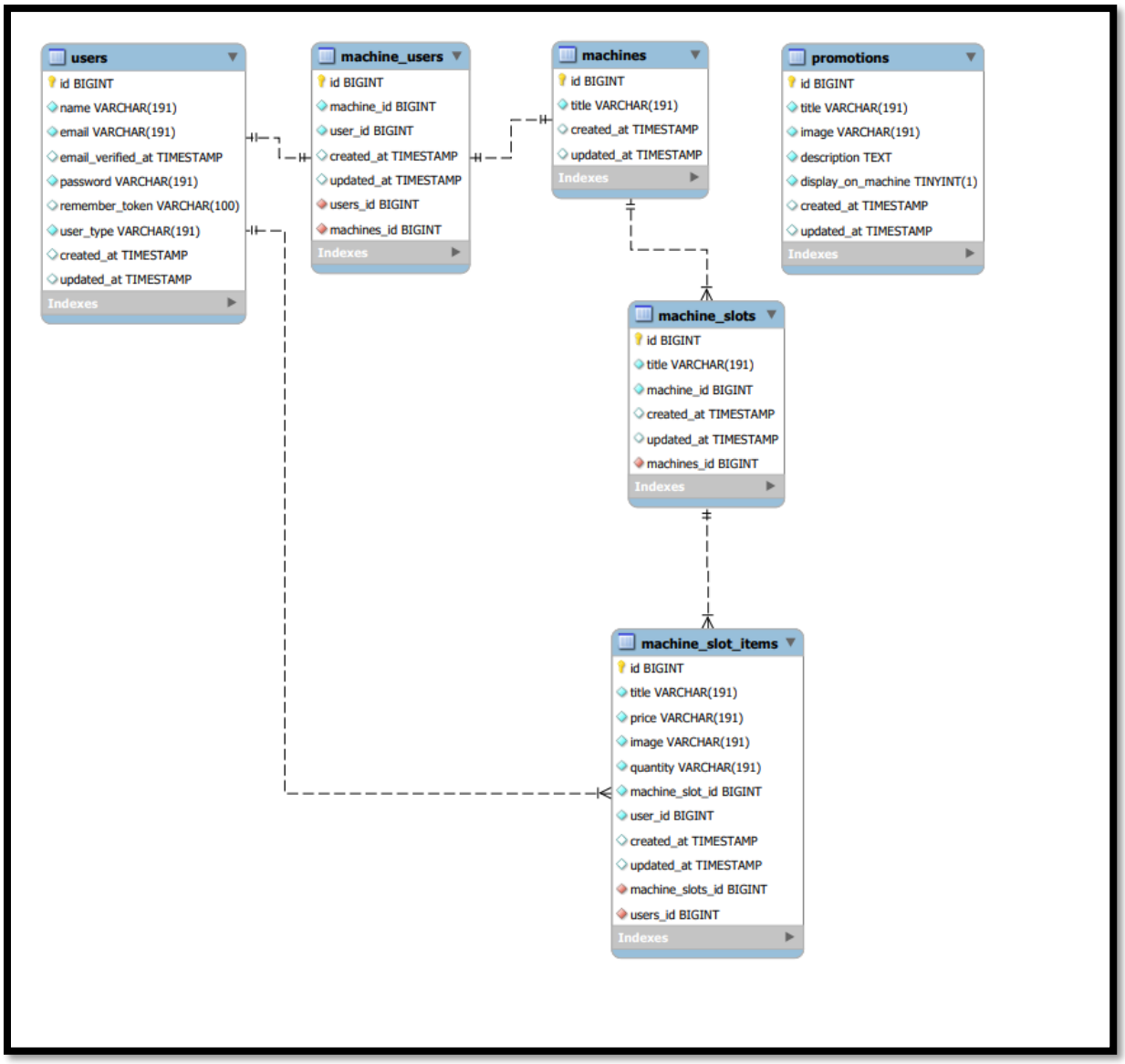

Figure 16: Database DFD

Pseudo Code :

Step 1: Start

Step 2: Define Namespace and Class for Console Kernel
```
namespace App\Console
class Kernel extends ConsoleKernel
```

Step 3: Define the `schedule` Method in Console Kernel
```
method schedule(schedule: Schedule)
    // $schedule->command('inspire')->hourly();
end method
```

Step 4: Define the `commands` Method in Console Kernel
```
method commands()
    loadCommands(__DIR__.'/Commands')
    requireRoutesConsole()
end method
```

Step 5: End Console Kernel Class
```
end class
```

Step 6: Start Web Routes

Step 7: Define Routes
```
Route::get('/', [HomeController::class, 'list'])->name('main');
Route::get('/machine/{id}', [HomeController::class, 'home'])->name('home');
Route::get('/product/{id}', [CartController::class, 'index'])->name('all-products');
Route::get('cart', [CartController::class, 'cart'])->name('cart');
Route::get('add-to-cart/{id}', [CartController::class, 'addToCart'])->name('add.to.cart');
Route::patch('update-cart', [CartController::class, 'update'])->name('update.cart');
Route::delete('remove-from-cart', [CartController::class, 'remove'])-
>name('remove.from.cart');
Route::get('promotions', [PromotionController::class, 'index'])->name('promotions-list');
Route::get('/checkout', [CartController::class, 'checkout'])->name('checkout');

Route::get('/admin', function () {
    return view('admin/login');
})->name('login');

Route::get('/seller', function () {
    return view('admin/login');
})->name('login');

Route::post('/admin', [AdminController::class, 'login'])->name('adminLogIn');
```

```
Route::middleware(['auth'])->prefix('admin')->group(function () {
    // Admin routes
});

Route::middleware(['auth'])->prefix('seller')->group(function () {
    // Seller routes
});
```

Step 8: End

Test Plan

- **Overall Test Case Table**

Description	Expected Result	Actual Outcome	Remarks
Enter Correct data in input field	Successfully Entered	Successfully Entered	**Pass**
Enter incorrect data in input field	Error result	Error result	**Pass**
Enter email correct and password incorrect	Error result	Error result	**Pass**
Enter password correct But email incorrect	Error result	Error result	**Pass**

Table 3: Test Case Table

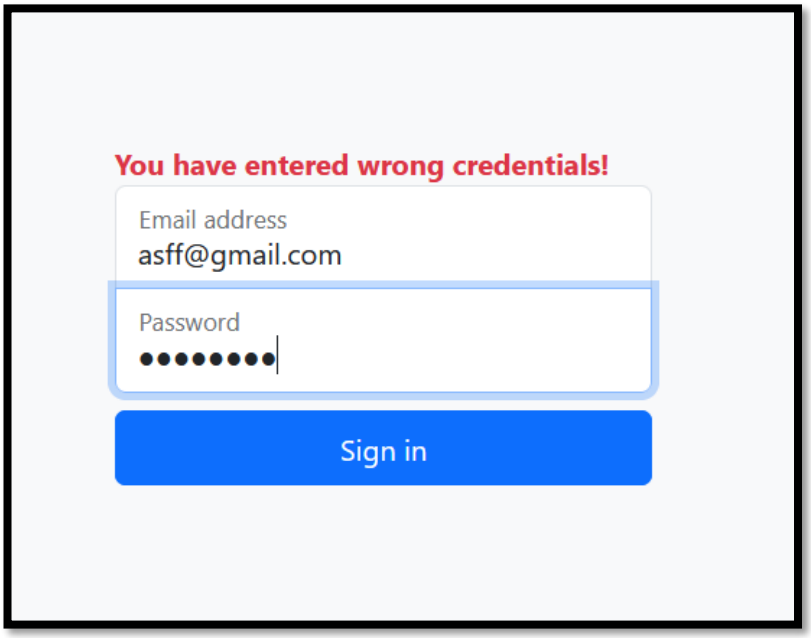

Table 4: Logging in with wrong credentials

Test Case	Description	Input	Expected Output
Test Case 1	Test the login method with valid credentials for an admin user	`'email' => 'admin@example.com', 'password' => 'admin123'`	Redirect to `/admin/dashboard` with success message
Test Case 2	Test the login method with valid credentials for a seller user	`'email' => 'seller@example.com', 'password' => 'seller123'`	Redirect to `/seller/machines` with success message
Test Case 3	Test the login method with invalid credentials	`'email' => 'invalid@example.com', 'password' => 'invalid'`	Redirect back with login error message
Test Case 4	Test the dashboard method	-	View rendered with the list of users
Test Case 5	Test the createUser method	-	View rendered for creating a new user
Test Case 6	Test the addUser method with valid input	`'name' => 'Test User', 'email' => 'testuser@example.com', 'password' => 'test123', 'user_type' => 1`	Redirect to `/admin/dashboard` with success message
Test Case 7	Test the addUser method with existing email	`'name' => 'Existing User', 'email' => 'admin@example.com', 'password' => 'existing123', 'user_type' => 1`	Redirect back with validation error
Test Case 8	Test the deleteUser method	`$id = 1`	Redirect to `/admin/dashboard` with success message
Test Case 9	Test the logout method	-	Redirect to `/admin`

Table 5: Test Case Table

Project Legacy

1. Current Status of Project:

As of the current date, the "Automat" project has successfully completed its development phase and is in the testing and deployment stage. The front-end and back-end components have been integrated, and the system is functional in a controlled environment. Testing is ongoing to ensure the system meets all specified requirements and performs seamlessly.

2. Remaining Areas of Concern:

While substantial progress has been made, a few areas of concern still require attention:

- Security Testing: Comprehensive security testing is crucial to identify and address any vulnerabilities in user authentication, data transactions, and system access.
- Performance Optimization: Further optimization is needed to enhance the loading speed of advertisements on the vending machine interface and ensure a smooth user experience.
- Usability Testing: Additional usability testing is required to gather user feedback and refine the user interfaces for both administrators and end-users.

3. Technical and Managerial Lessons Learned:

- Collaborative Development: The multidisciplinary team structure proved effective in addressing various aspects of the project. Regular communication and collaboration enhanced problem-solving and decision-making.
- Technology Stack Selection: The choice of HTML, CSS, PHP, Laravel, and MySQL as the technology stack provided a solid foundation for development, offering both flexibility and scalability.
- Agile Methodology: The adoption of an agile development methodology facilitated iterative progress, allowing for adjustments based on ongoing feedback and changing requirements.
- User-Centric Design: Emphasizing user-centric design principles in the development process contributed to a more intuitive and engaging user interface.

4. Future Recommendations:

- Security Enhancements: Implement additional security measures, such as encryption for sensitive data and regular security audits, to fortify the system against potential threats.
- Performance Tuning: Continue refining performance optimization strategies to ensure fast and efficient loading of advertisements and responsive user interfaces.
- User Feedback Integration: Encourage user feedback through usability testing and incorporate insights to refine the user interface and overall user experience.
- Scalability Planning: As the project progresses, consider scalability factors to accommodate potential increases in user traffic and data volume.
- Documentation: Maintain comprehensive documentation for the entire system, including codebase, configurations, and deployment procedures, to facilitate future maintenance and updates.

Conclusion:

The current status reflects a well-developed system with ongoing efforts to address remaining concerns. Technical and managerial lessons learned provide valuable insights for future projects, and the outlined recommendations guide the project's evolution beyond the current stage. The "Automat" project aims to set new standards in vending machine management systems through continuous improvement and adherence to best practices.

Bibliography

1. https://github.com/UplandsDynamic/simple-stock-management
2. https://github.com/gavindsouza/inventory-management-system
3. https://github.com/AsjadIqbal/InventoryManagementSystem
4. https://github.com/hassanrazadev/LaravelInventoryWithPOS
5. https://github.com/LeftTwixWand/Inventory
6. https://github.com/ekramasif/Inventory_Management_System
7. https://github.com/Akash-goyal-github/Inventory-Management-System
8. https://github.com/w-a-r-m-inventory-system/Food-Pantry-Inventory
9. https://www.sanfoundry.com/best-reference-books-inventory-control-management-systems/#inventory-control-management-systems
10. https://www.amazon.com/Inventory-Management-Software-Complete-Self-Assessment/dp/0655189645
11. Inventory Management Explained: A Focus on Forecasting, Lot Sizing, Safety Stock, and Ordering Systems: By David J. Piasecki, 2009
12. https://github.com/anantjain6/ProductManagementSystem
13. https://github.com/LieLieLiekey/Small-supermarket-inventory-management-system
14. https://github.com/code-boxx/Storage-Boxx-PHP-Inventory-System
15. https://github.com/AsjadIqbal/InventoryManagementSystem
16. https://github.com/hassanrazadev/LaravelInventoryWithPOS
17. https://github.com/LeftTwixWand/Inventory
18. https://github.com/ekramasif/Inventory_Management_System
19. https://www.sanfoundry.com/best-reference-books-inventory-control-management-systems/#inventory-control-management-systems
20. https://www.amazon.com/Inventory-Management-Software-Complete-Self-Assessment/dp/0655189645

Source Code

Note:

1. If we add all file code then it will take more space so we add only main file code.
2. But all the files are uploaded on Git-Hub and the URL are attached below: https://github.com/devgupta55/majorProject

AutoMat file source code:

```
<?php

use App\Http\Controllers\Admin\AdminController;
use App\Http\Controllers\Admin\MachineController;
use App\Http\Controllers\Admin\PromotionController;
use App\Http\Controllers\CartController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\Seller\SellerController;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "web" middleware group. Make something great!
|
*/

Route::get('/', [HomeController::class, 'list'])->name('main');
Route::get('/machine/{id}', [HomeController::class, 'home'])->name('home');

Route::get('/product/{id}', [CartController::class, 'index'])->name('all-
products');
Route::get('cart', [CartController::class, 'cart'])->name('cart');
Route::get('add-to-cart/{id}', [CartController::class, 'addToCart'])-
>name('add.to.cart');
Route::patch('update-cart', [CartController::class, 'update'])-
>name('update.cart');
Route::delete('remove-from-cart', [CartController::class, 'remove'])-
>name('remove.from.cart');
Route::get('promotions', [PromotionController::class, 'index'])-
>name('promotions-list');
Route::get('/checkout', [CartController::class, 'checkout'])-
>name('checkout');
```

```
Route::get('/admin', function () {
    return view('admin/login');
})->name('login');

Route::get('/seller', function () {
    return view('admin/login');
})->name('login');

Route::post('/admin', [AdminController::class, 'login'])->name('adminLogIn');

Route::middleware(['auth'])->prefix('admin')->group(function () {
    Route::get('/dashboard', [AdminController::class, 'dashBoard'])-
>name('dasboard');
    Route::get('/logout', [AdminController::class, 'logout'])->name('admin-
logout');
    Route::get('/user', [AdminController::class, 'createUser'])->name('create-
user');
    Route::post('/user', [AdminController::class, 'addUser'])->name('add-
user');
    Route::get('/user/delete/{id}', [AdminController::class, 'deleteUser'])-
>name('delete-user');

    Route::get('/machines', [MachineController::class, 'listing'])-
>name('machines');
    Route::get('/machines/create', [MachineController::class, 'create'])-
>name('create-machine');
    Route::post('/machines/create', [MachineController::class, 'addMachine'])-
>name('add-machine');
    Route::get('/machines/delete/{id}', [MachineController::class, 'delete'])-
>name('delete-machine');

    Route::get('/machines/assign/{machineId}', [MachineController::class,
'assign'])->name('assign-machine');
    Route::post('/machines/assign/{machineId}', [MachineController::class,
'assignUser'])->name('assign-machines-user');

    Route::get('/machines/slots/{machineId}', [MachineController::class,
'slots'])->name('slots');
    Route::get('/machines/slots/create/{machineId}',
[MachineController::class, 'createSlot'])->name('create-machine-slot');
    Route::get('/machines/slots/delete/{id}', [MachineController::class,
'deleteSlot'])->name('delete-machine-slot');
    Route::post('/machines/slots/create', [MachineController::class,
'addSlot'])->name('add-machine-slot');

    Route::get('/machines/slots/item/{slotId}', [MachineController::class,
'slotItems'])->name('slot-item');
```

```
    Route::get('/machines/slots/item/create/{slotId}',
[MachineController::class, 'createSlotItem'])->name('create-machine-slot-
item');
    Route::post('/machines/slots/item/create/{slotId}',
[MachineController::class, 'addSlotItem'])->name('add-machine-slot-item');
    Route::post('/machines/slots/item/delete/{id}', [MachineController::class,
'addSlotItem'])->name('delete-machine-slot-item');


    Route::get('/promotions', [PromotionController::class, 'listing'])-
>name('promotions');
    Route::get('/promotions/create', [PromotionController::class, 'create'])-
>name('create-promotions');
    Route::get('/promotions/edit/{id}', [PromotionController::class, 'edit'])-
>name('edit-promotions');
    Route::post('/promotions/update/{id}', [PromotionController::class,
'update'])->name('update-promotions');
    Route::post('/promotions/create', [PromotionController::class, 'add'])-
>name('add-promotions');
    Route::get('/promotions/delete/{id}', [PromotionController::class,
'delete'])->name('delete-promotions');
});

Route::middleware(['auth'])->prefix('seller')->group(function () {
    Route::get('/machines', [SellerController::class, 'sellerMachines'])-
>name('seller-machines');
    Route::get('/machines/slots/{machineId}', [SellerController::class,
'slots'])->name('seller-slots');

    Route::get('/machines/slots/item/{slotId}', [SellerController::class,
'slotItems'])->name('seller-slot-item');
    Route::get('/machines/slots/item/create/{slotId}',
[SellerController::class, 'createSlotItem'])->name('create-machine-slot-item-
seller');
    Route::post('/machines/slots/item/create/{slotId}',
[SellerController::class, 'addSlotItem'])->name('add-machine-slot-item-
seller');
    Route::get('/machines/slots/item/edit/{slotId}', [SellerController::class,
'slots'])->name('edit-slot-item');
    Route::post('/machines/slots/item/edit/{slotId}',
[SellerController::class, 'slots'])->name('update-slot-item');
    Route::post('/machines/slots/item/delete/{slotId}',
[SellerController::class, 'deleteSlotItem'])->name('delete-slot-item');

});
```

JS Code

```
/**
 * We'll load the axios HTTP library which allows us to easily issue requests
 * to our Laravel back-end. This library automatically handles sending the
 * CSRF token as a header based on the value of the "XSRF" token cookie.
 */

import axios from 'axios';
window.axios = axios;

window.axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';

/**
 * Echo exposes an expressive API for subscribing to channels and listening
 * for events that are broadcast by Laravel. Echo and event broadcasting
 * allows your team to easily build robust real-time web applications.
 */

// import Echo from 'laravel-echo';

// import Pusher from 'pusher-js';
// window.Pusher = Pusher;

// window.Echo = new Echo({
//     broadcaster: 'pusher',
//     key: import.meta.env.VITE_PUSHER_APP_KEY,
//     cluster: import.meta.env.VITE_PUSHER_APP_CLUSTER ?? 'mt1',
//     wsHost: import.meta.env.VITE_PUSHER_HOST ?
import.meta.env.VITE_PUSHER_HOST : `ws-
${import.meta.env.VITE_PUSHER_APP_CLUSTER}.pusher.com`,
//     wsPort: import.meta.env.VITE_PUSHER_PORT ?? 80,
//     wssPort: import.meta.env.VITE_PUSHER_PORT ?? 443,
//     forceTLS: (import.meta.env.VITE_PUSHER_SCHEME ?? 'https') === 'https',
//     enabledTransports: ['ws', 'wss'],
// });
```

Index File

```
<?php

use Illuminate\Contracts\Http\Kernel;
use Illuminate\Http\Request;
```

```php
define('LARAVEL_START', microtime(true));

/*
|--------------------------------------------------------------------------
| Check If The Application Is Under Maintenance
|--------------------------------------------------------------------------
|
| If the application is in maintenance / demo mode via the "down" command
| we will load this file so that any pre-rendered content can be shown
| instead of starting the framework, which could cause an exception.
|
*/

if (file_exists($maintenance =
__DIR__.'/../storage/framework/maintenance.php')) {
    require $maintenance;
}

/*
|--------------------------------------------------------------------------
| Register The Auto Loader
|--------------------------------------------------------------------------
|
| Composer provides a convenient, automatically generated class loader for
| this application. We just need to utilize it! We'll simply require it
| into the script here so we don't need to manually load our classes.
|
*/

require __DIR__.'/../vendor/autoload.php';

/*
|--------------------------------------------------------------------------
| Run The Application
|--------------------------------------------------------------------------
|
| Once we have the application, we can handle the incoming request using
| the application's HTTP kernel. Then, we will send the response back
| to this client's browser, allowing them to enjoy our application.
|
*/

$app = require_once __DIR__.'/../bootstrap/app.php';

$kernel = $app->make(Kernel::class);

$response = $kernel->handle(
    $request = Request::capture()
```

```
)->send();

$kernel->terminate($request, $response);
```

CSS

```
.bi {
  display: inline-block;
  width: 1rem;
  height: 1rem;
}

/*
 * Sidebar
 */

@media (min-width: 768px) {
  .sidebar .offcanvas-lg {
    position: -webkit-sticky;
    position: sticky;
    top: 48px;
  }
  .navbar-search {
    display: block;
  }
}

.sidebar .nav-link {
  font-size: .875rem;
  font-weight: 500;
}

.sidebar .nav-link.active {
  color: #2470dc;
}

.sidebar-heading {
  font-size: .75rem;
}

/*
 * Navbar
 */

.navbar-brand {
  padding-top: .75rem;
  padding-bottom: .75rem;
```

```
  background-color: rgba(0, 0, 0, .25);
  box-shadow: inset -1px 0 0 rgba(0, 0, 0, .25);
}

.navbar .form-control {
  padding: .75rem 1rem;
}
```

CSS 2

```
html,
body {
  height: 100%;
}

.form-signin {
  max-width: 330px;
  padding: 1rem;
}

.form-signin .form-floating:focus-within {
  z-index: 2;
}

.form-signin input[type="email"] {
  margin-bottom: -1px;
  border-bottom-right-radius: 0;
  border-bottom-left-radius: 0;
}

.form-signin input[type="password"] {
  margin-bottom: 10px;
  border-top-left-radius: 0;
  border-top-right-radius: 0;
}
```

CSS 3

```
.thumbnail {
    position: relative;
    padding: 0px;
    margin-bottom: 20px;
}
.thumbnail img {
    width: 80%;
}
```

```
.thumbnail .caption{
    margin: 7px;
}
.main-section{
    background-color: #F8F8F8;
}
.dropdown{
    float:right;
    padding-right: 30px;
}
/* .btn{
    border:0px;
    margin:10px 0px;
    box-shadow:none !important;
} */
.dropdown .dropdown-menu{
    padding:20px;
    top:55px !important;
    width:350px !important;
    left:-200px !important;
    box-shadow:0px 5px 30px black;
}
.total-header-section{
    border-bottom:1px solid #d2d2d2;
}
.total-section p{
    margin-bottom:20px;
}
.cart-detail{
    padding:15px 0px;
}
.cart-detail-img img{
    width:100%;
    height:100%;
    padding-left:15px;
}
.cart-detail-product p{
    margin:0px;
    color:#000;
    font-weight:500;
}
.cart-detail .price{
    font-size:12px;
    margin-right:10px;
    font-weight:500;
}
.cart-detail .count{
    color:#C2C2DC;
```

```
}
.checkout{
    border-top:1px solid #d2d2d2;
    padding-top: 15px;
}
.checkout .btn-primary{
    border-radius:50px;
    height:50px;
}
.dropdown-menu:before{
    content: " ";
    position:absolute;
    top:-20px;
    right:50px;
    border:10px solid transparent;
    border-bottom-color:#fff;
}
```