Problem Statement:

The ice cream shop faces challenges in managing its sales data effectively. Currently, the staff manually records sales transactions, a process that is not only time-consuming but also prone to human error. Without an automated system, it's difficult for the shop's management to quickly access or analyse this data, leading to delays in decision-making and potential losses in revenue.

Furthermore, the manual system lacks the capability to provide insights into key business metrics such as which products are selling best, how sales fluctuate over different periods, and identifying sales trends. This deficiency hampers the management's ability to make informed decisions regarding inventory control, promotional strategies, and overall sales tactics.

The absence of real-time data analysis means the shop cannot adjust quickly to changing customer preferences or operational challenges. For instance, if a particular flavour of ice cream is selling exceptionally well or poorly, the current system does not support swift analysis and response, potentially leading to either stock shortages or overages, both of which are costly to the business.

The goal, therefore, is to develop a system that not only automates the recording and storage of sales data but also analyses this data to provide actionable insights in an easy-to-understand format. This system should reduce the workload on staff, decrease the likelihood of errors, and enable the management to access and interpret sales data promptly and accurately.

Proposed System:

The proposed system is designed to be a robust, web-based application specifically tailored for an ice cream shop to enhance its operational and strategic capabilities. This system will replace the current manual processes with an automated, digital solution that not only records transactions but also provides a comprehensive suite of analytical tools. Here’s a detailed breakdown of the system features:

Real-time Sales Tracking

This feature automates the recording of every sale as it happens, ensuring that the data is up-to-date and accurate. The system captures essential details such as the product sold, the quantity, the sale price, and the time of transaction. This automation reduces human error and frees up staff time, allowing them to focus more on customer service rather than manual record-keeping.

Analytics Dashboard

The dashboard is the heart of the system where data is transformed into easy-to-understand graphs and charts. It displays sales performance over different periods—daily, weekly, monthly, and yearly—giving the management clear visibility into sales trends. This feature helps in quickly identifying patterns, assessing business performance over time, and making informed decisions based on current trends.

User Authentication

Security is a priority, especially when handling sales data. The system includes a secure login mechanism that ensures only authorized personnel have access to the analytics dashboard. This feature protects sensitive business information and maintains data integrity.

Product Performance Analysis

By analyzing the sales data, the system identifies which products are performing well and which are not. This feature helps the shop in inventory management by understanding which items to stock more based-on popularity and sales performance. It also aids in making strategic decisions such as which products to promote more and which to phase out or run promotional offers on.

Create New Order/Invoice Page

A key feature of the system is the ability to create new orders and generate invoices directly from the interface. When a customer places an order, the staff can quickly enter the details into the system, which automatically calculates the total cost, adds

applicable taxes, and generates an invoice. This process not only speeds up the transaction but also ensures accuracy in billing. Once the transaction is complete, the order details are immediately recorded in the system, contributing to the real-time sales data.

Implementation Benefits

Implementing this proposed system will bring numerous benefits to the ice cream shop:

- **Efficiency:** Automating sales tracking and invoice generation speeds up operations and reduces manual tasks.
- **Accuracy:** Minimizes the risk of human errors in sales recording and financial calculations.
- **Insights:** Provides valuable insights into business performance, helping to drive strategic decisions and improve profitability.
- **Data Security:** Ensures that sensitive business information is protected and only accessible to authorized personnel.

This system is expected to transform how the ice cream shop manages its operations, making it more efficient, data-driven, and responsive to the dynamic market environment.

Hardware and Software Requirements:

Hardware:

- **Server:** Minimum Intel Core i5 or equivalent, 8GB RAM, and 100GB HDD.
- **Client Devices:** Computers or tablets with internet access for accessing the web application.

Software:

- **Operating System:** Compatible with all major operating systems including Windows, macOS, and Linux, facilitating broad accessibility.

- **Backend Frameworks: Python 3.8+ :**
    - **Flask:** A lightweight WSGI web application framework used to handle requests and serve data to the front end.
    - **Flask-SQLAlchemy:** Provides ORM support for Flask applications, simplifying database management and interactions.
    - **mysql-connector-python:** A MySQL driver used for connecting the Flask application to a MySQL database.
    - **Flask-Login:** Manages user authentication, making it easy to handle logins, logouts, and session management.
    - **Flask-Bcrypt:** Used for hashing passwords to enhance security measures within the application**.**
    - **python-dotenv:** Handles application configuration; allows storing configuration variables in a .env file, improving security and flexibility.
- **Database: MySQL:** Utilized for storing all application data including user credentials, transaction records, and analytics data.
- **Frontend:**
    - **HTML5/CSS3:** Standard technologies for building web interfaces, ensuring compatibility and responsiveness.
    - **JavaScript:** Powers interactive features on the web pages, enhancing user experience.
    - **Bootstrap:** A framework used for developing responsive and mobile-first web pages.
    - **Google Charts:** Integrated for dynamic data visualization, transforming raw data into understandable charts and graphs.

ER Diagram:

Understanding ER Diagrams

An Entity-Relationship (ER) Diagram is a specialized graphic that illustrates the relationships between entities in a database. ER diagrams are essential for

modelling and designing databases; they help to systematically represent data structures and their interconnections, which facilitates building efficient and error-free databases. This visualization is crucial during the initial phases of a database design - allowing database designers to conceptualize the structure and relationships of data before actual implementation.

Current Database Design

The proposed system for the ice cream shop utilizes a database with four main entities: **User**, **Product**, **Order**, and **OrderItem**. Here's a detailed breakdown of each entity and their relationships:

- **User**: Represents the staff or administrators who can interact with the system. Each user has a unique identifier, a username, and a password hash stored for authentication.
    - Attributes: id, username, password_hash
- **Product**: Represents items available for sale (e.g., different types of ice cream). Each product is characterized by an identifier, name, size, and price.
    - Attributes: id, name, size, price
- **Order**: Represents a transaction or a sale. It includes details like the total amount before and after taxes, the GST amount, and a timestamp of when the order was placed. Each order is associated with a specific user.
    - Attributes: id, user_id, total_amount, gst_amount, final_amount, order_time, status
- **OrderItem**: Represents individual items within an order. This entity links products to their respective orders, detailing the quantity of each product ordered and the price at the time of the order.
    - Attributes: id, order_id, product_id, quantity, item_price

Relationships:

- **User to Order**: One-to-Many. A user can place multiple orders, but each order is linked to only one user.
- **Order to OrderItem**: One-to-Many. An order can consist of multiple order items.

- **Product to OrderItem**: One-to-Many. A product can be part of multiple order items across different orders.

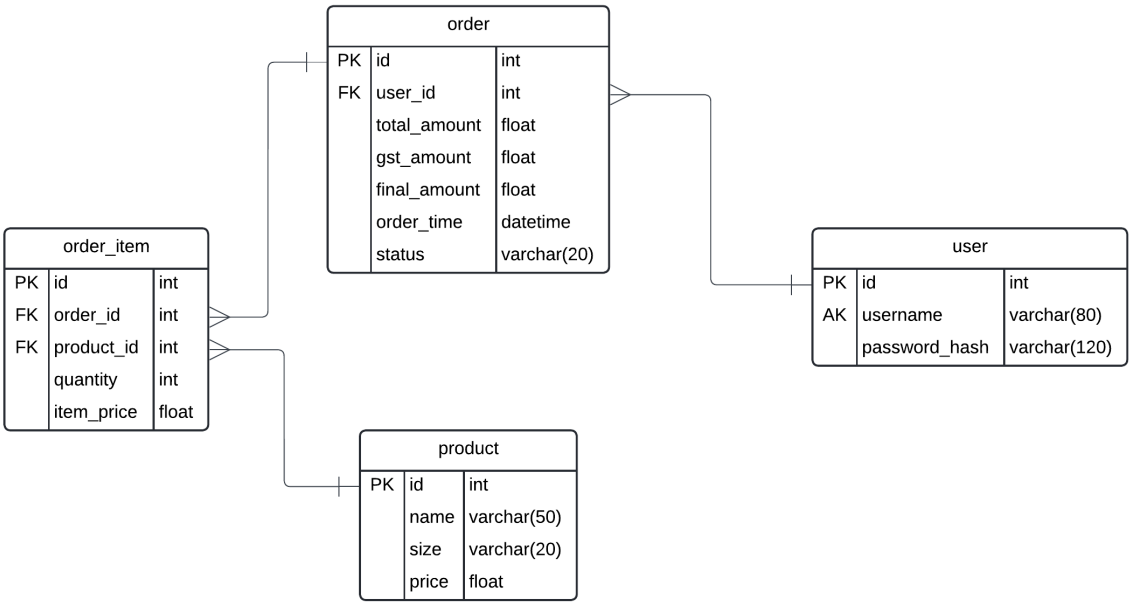
Schema Details:

Each entity is defined with specific fields that store data relevant to the application's functionality:

- **User**: Stores login credentials and links to orders placed by the user.
- **Product**: Maintains a catalog of items that can be ordered.
- **Order**: Records details about sales transactions, including references to the users and the computed totals.
- **OrderItem**: Connects products to their orders, specifying quantities and prices, which are essential for inventory and sales analysis.

Visual Representation:

To aid in visualizing this structure, an ER diagram would typically show these entities as rectangles connected by lines that represent their relationships. Each relationship line would be annotated to describe the nature of the relationship (one-to-many, many-to-one), and cardinality markers would indicate the optional or mandatory nature of the relationships.



ER Diagram for Ice Cream Shop

This ER diagram effectively maps out how data flows between different parts of the system, ensuring that the database supports all required operations of the sales analytics platform efficiently.

Database Models and SQL Code:

SQLAlchemy Database Models

SQLAlchemy ORM is used to define the database schema in Python, providing a class-based format that simplifies database interactions and enhances code maintainability. It enables easy management of database relationships and reduces SQL injection risks through ORM methods.

```python
from app import db, app, bcrypt
import datetime
from flask_login import UserMixin


class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password_hash = db.Column(db.String(120), nullable=False)

    orders = db.relationship("Order", backref="user", lazy=True)

    def set_password(self, password):
        self.password_hash = bcrypt.generate_password_hash(password)

    def check_password(self, password):
        return bcrypt.check_password_hash(self.password_hash, password)


class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    size = db.Column(db.String(20), nullable=False)
    price = db.Column(db.Float, nullable=False)

    order_items = db.relationship("OrderItem", backref="product", lazy=True)


class Order(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"), nullable=False)
    total_amount = db.Column(db.Float)
    gst_amount = db.Column(db.Float)
    final_amount = db.Column(db.Float)
    order_time = db.Column(
        db.DateTime, default=datetime.datetime.now(datetime.UTC)
    )
    status = db.Column(db.String(20), default="Pending")

    items = db.relationship("OrderItem", backref="order", lazy=True)


class OrderItem(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    order_id = db.Column(
        db.Integer, db.ForeignKey("order.id"), nullable=False
    )
    product_id = db.Column(
        db.Integer, db.ForeignKey("product.id"), nullable=False
    )
    quantity = db.Column(db.Integer, nullable=False)
    item_price = db.Column(db.Float, nullable=False)


with app.app_context():
    db.create_all()

```

MySQL Database Schema

The MySQL database schema outlines the direct structure of tables, including keys, data types, and constraints, essential for database setup in MySQL. This schema ensures accurate data management and enforces relationships and integrity within the database system.

```
-- MySQL dump 10.13  Distrib 8.3.0, for macos14.2 (arm64)
--
-- Host: localhost    Database: icecreamshop
-- ------------------------------------------------------
-- Server version 8.3.0

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!50503 SET NAMES utf8mb4 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0
*/;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `order`
--

DROP TABLE IF EXISTS `order`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `order` (
  `id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `total_amount` float DEFAULT NULL,
  `gst_amount` float DEFAULT NULL,
  `final_amount` float DEFAULT NULL,
  `order_time` datetime DEFAULT NULL,
```

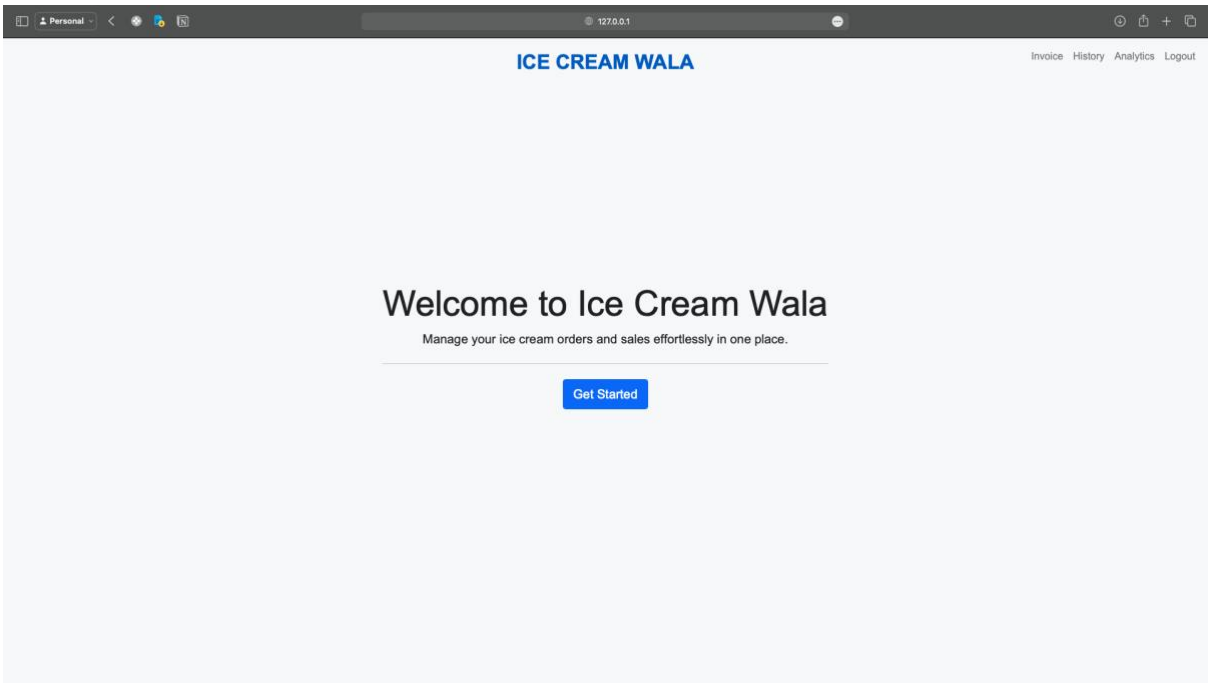
```
  `status` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `user_id` (`user_id`),
  CONSTRAINT `order_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=99 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;



--
-- Table structure for table `order_item`
--

DROP TABLE IF EXISTS `order_item`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `order_item` (
  `id` int NOT NULL AUTO_INCREMENT,
  `order_id` int NOT NULL,
  `product_id` int NOT NULL,
  `quantity` int NOT NULL,
  `item_price` float NOT NULL,
  PRIMARY KEY (`id`),
  KEY `order_id` (`order_id`),
  KEY `product_id` (`product_id`),
  CONSTRAINT `order_item_ibfk_1` FOREIGN KEY (`order_id`) REFERENCES `order` (`id`),
  CONSTRAINT `order_item_ibfk_2` FOREIGN KEY (`product_id`) REFERENCES `product` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=254 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;


--
-- Table structure for table `product`
--

DROP TABLE IF EXISTS `product`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `product` (
  `id` int NOT NULL AUTO_INCREMENT,
```

```
  `name` varchar(50) NOT NULL,
  `size` varchar(20) NOT NULL,
  `price` float NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;



--
-- Table structure for table `user`
--

DROP TABLE IF EXISTS `user`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `user` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(80) NOT NULL,
  `password_hash` varchar(120) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;


/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2024-05-07 10:18:19
```

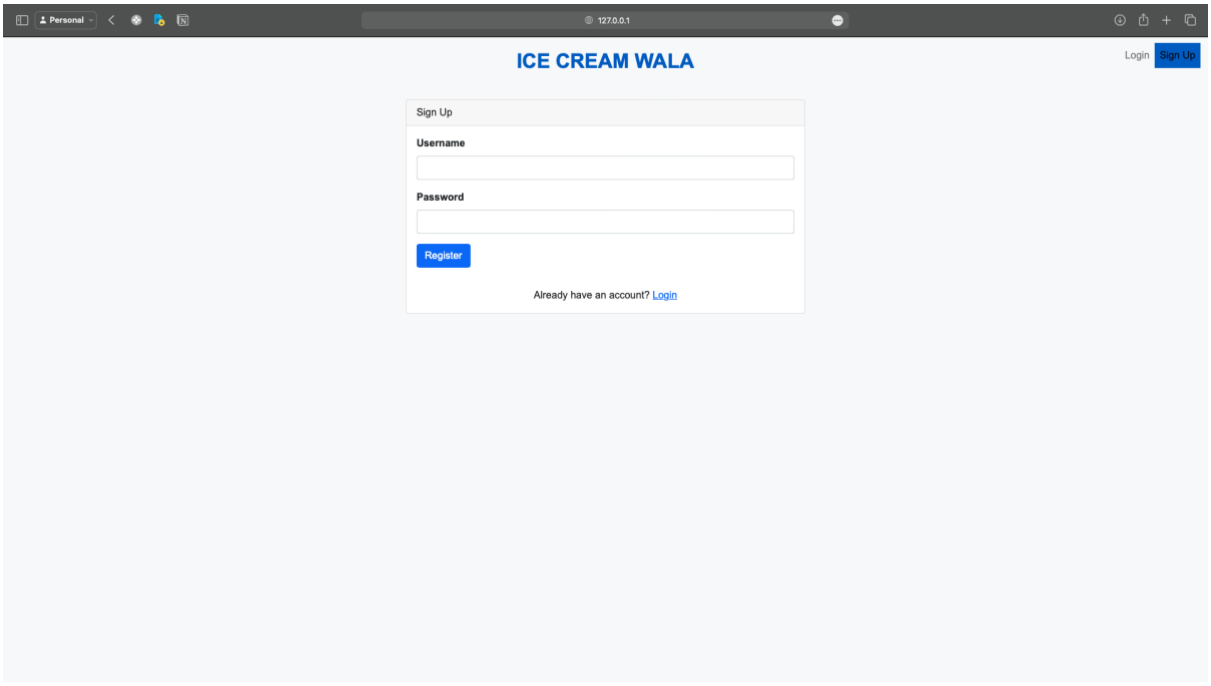
Results:

Our comprehensive sales management system, **Ice Cream Shop Insights**, effectively enhances operational efficiency and oversight through a meticulously designed, feature-rich platform tailored to the unique needs of an ice cream shop. This system automates and simplifies the task of sales tracking and analysis through the following key features:

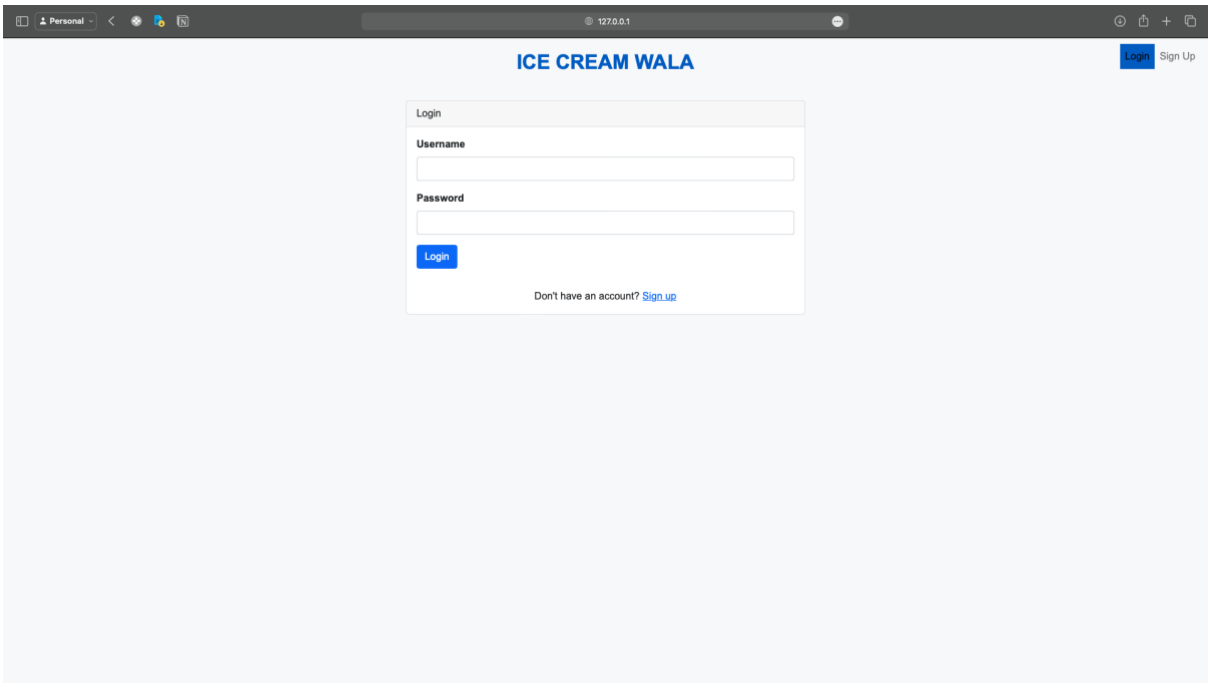
1. **Home Page (Landing Page)**: Provides an introduction to the application, highlighting key features and benefits, which offers a welcoming first impression to users.



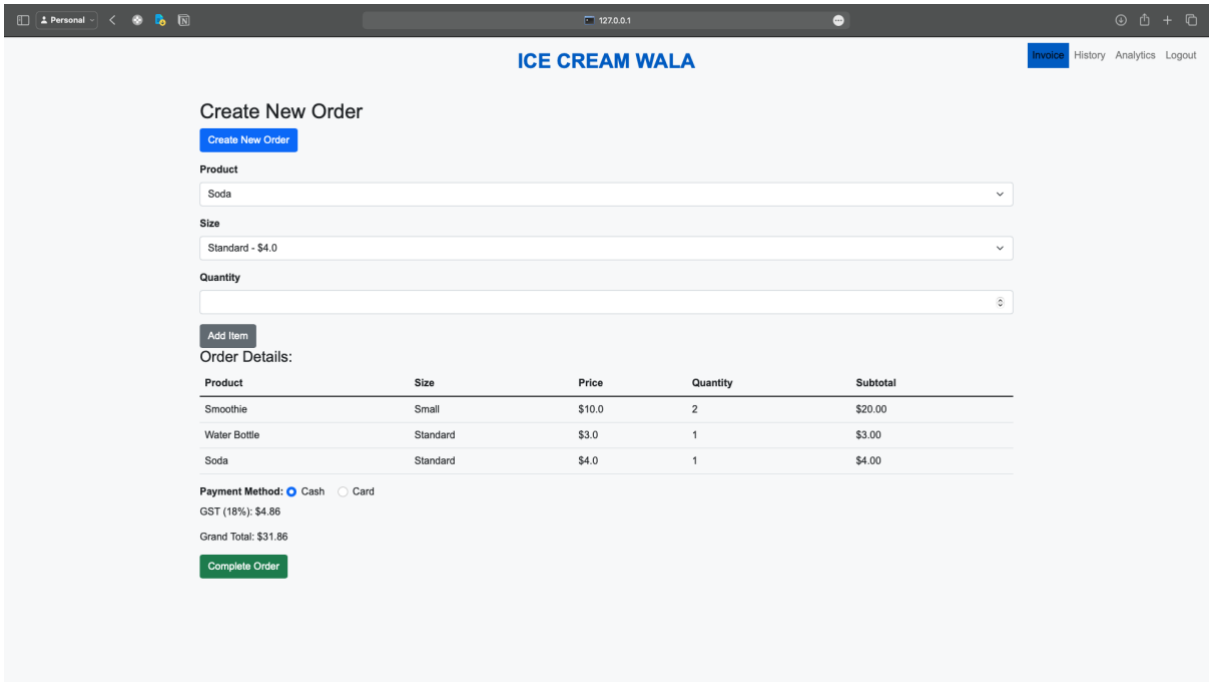
2. **Admin Registration (Signup Page)**: Administrators can sign up securely, establishing controlled access to manage and oversee the system.



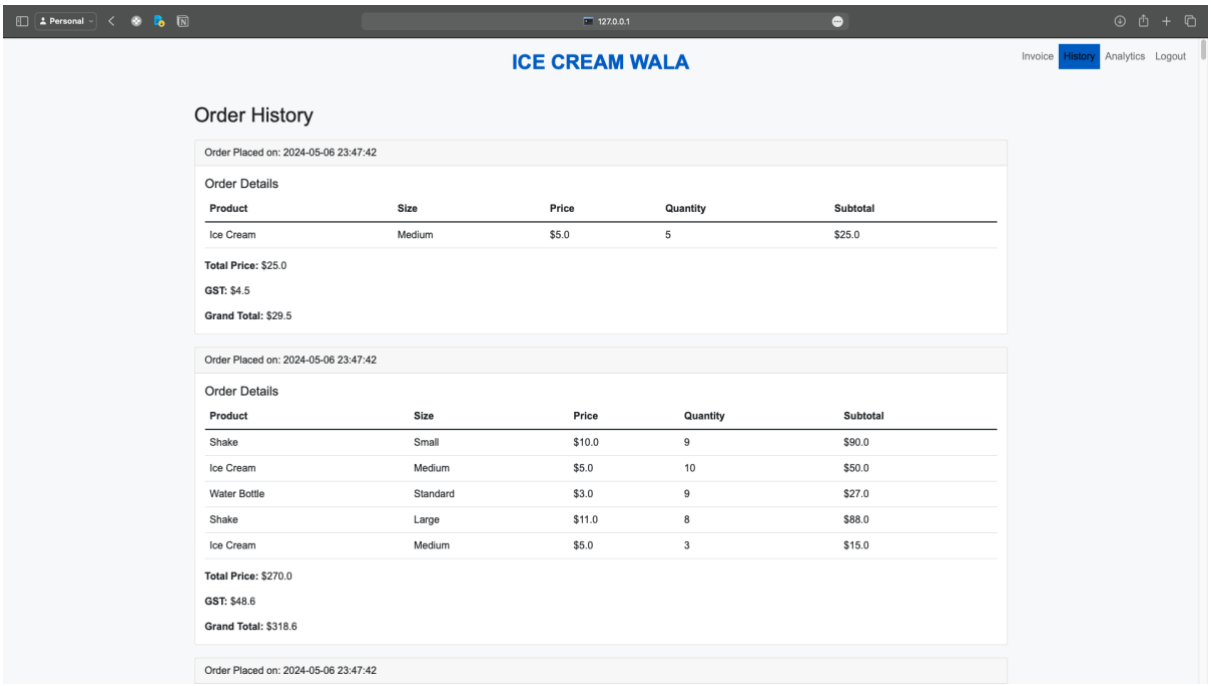
3. **Secure Login**: Users log in through a secure portal, ensuring their information is protected while accessing their personalized dashboard.



4. **Invoice Creation (Invoice Page)**: Enables users to create new orders, add items, calculate totals with GST, and complete transactions seamlessly, updating the system in real-time.

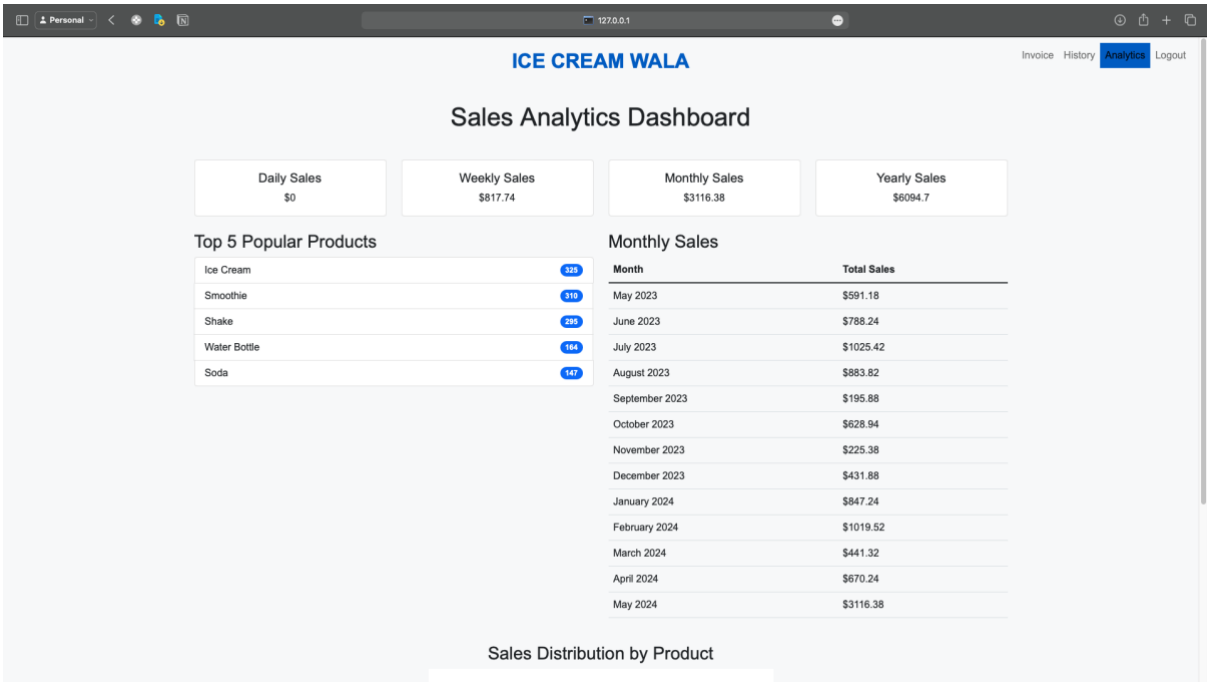


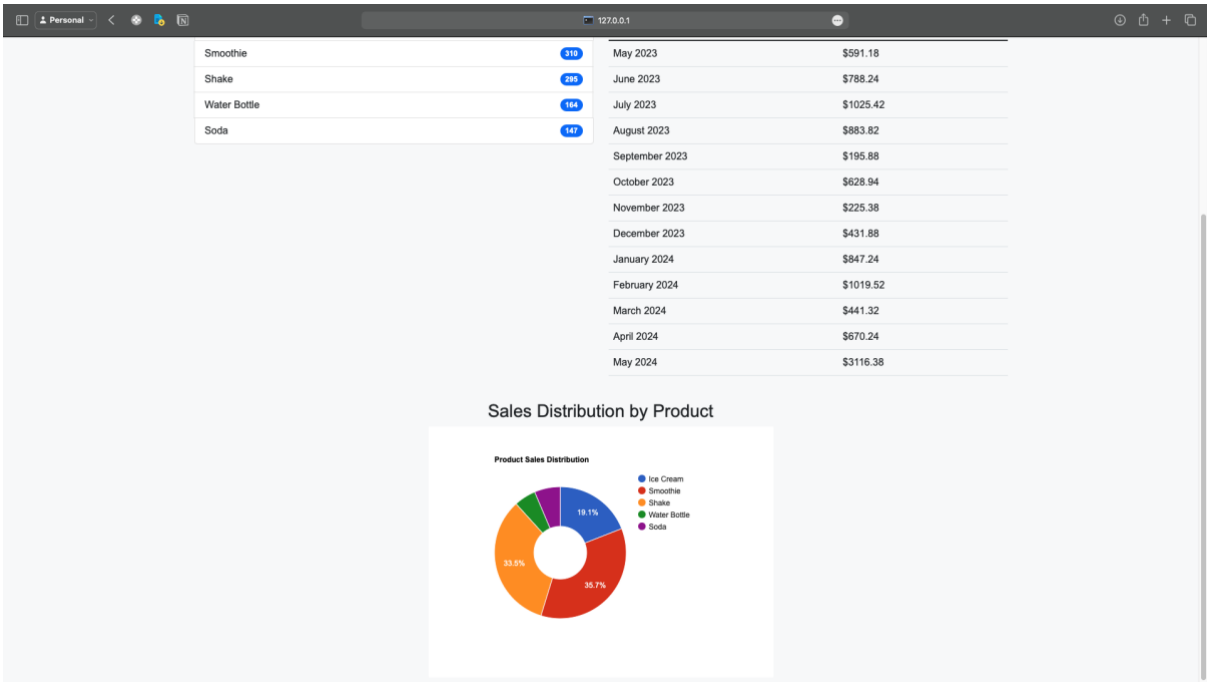
5. **Order History (History Page)**: Provides a detailed log of past transactions, allowing users to view and analyse previous sales data.



6. **Comprehensive Analytics Dashboard**:
    - **Daily, Weekly, Monthly, and Yearly Sales**: Displays sales data in various intervals for precise tracking and trend analysis.

- **Top 5 Popular Products**: Highlights the best-selling items to inform stock and marketing decisions.
- **Monthly Sales Breakdown**: Offers detailed insights into sales performance month-over-month.
- **Sales Distribution by Product**: Visualizes product performance across different categories, aiding in product strategy adjustments.





Conclusion:

The **Ice Cream Shop Insights** system has successfully met its objectives to streamline and enhance the sales management process within the ice cream shop environment. This tool combines extensive functionality with user-friendly interfaces, fostering an integrated approach to business management. Key outcomes include:

- **Increased Operational Efficiency**: Automates complex processes, significantly reducing manual effort and increasing accuracy.
- **Enhanced Strategic Insight**: Provides detailed analytics that support dynamic decision-making and help optimize operational strategies.
- **Improved Product Management**: Enables precise tracking of product performance, which assists in inventory and marketing strategies.
- **Robust Security and Scalability**: Ensures data integrity and privacy with advanced security measures, designed to scale seamlessly with business growth.

In essence, **Ice Cream Shop Insights** is not just a tool but a pivotal element in transforming everyday business operations into strategic, data-driven processes. This system is instrumental in pushing the boundaries of traditional sales management, paving the way for enhanced productivity and profitability in a competitive retail landscape.

Reference:

1. Flask-Login Documentation: Flask-Login. (n.d.). Flask-Login Documentation. Retrieved December 11, 2023, from https://flask-login.readthedocs.io/en/latest/
2. Flask-SQLAlchemy Documentation: Pallets Projects. (n.d.). Flask-SQLAlchemy Documentation. Retrieved December 11, 2023, from https://flask-sqlalchemy.palletsprojects.com/en/3.1.x/

3. Flask-Bcrypt Documentation: Flask-Bcrypt. (n.d.). Flask-Bcrypt Documentation. Retrieved December 11, 2023, from https://flask-bcrypt.readthedocs.io/en/1.0.1/

4. Python-dotenv Documentation: Python-dotenv. (n.d.). Python-dotenv Documentation. Retrieved December 11, 2023, from https://pypi.org/project/python-dotenv/

5. **mysql-connector-python Documentation**: Oracle. (n.d.). MySQL Connector/Python Developer Guide. Retrieved December 11, 2023, from https://dev.mysql.com/doc/connector-python/en/
6. **Flask Documentation**: Pallets Projects. (n.d.). Flask Documentation. Retrieved December 11, 2023, from https://flask.palletsprojects.com/en/2.2.x/
7. **Bootstrap Documentation**: Bootstrap. (n.d.). Bootstrap Documentation. Retrieved December 11, 2023, from https://getbootstrap.com/docs/5.2/getting-started/introduction/
8. **Google Charts Documentation**: Google Charts. (n.d.). Google Charts Documentation. Retrieved December 11, 2023, from https://developers.google.com/chart

APPENDIX:

/app/routes.py

```
from datetime import datetime, timedelta, date
from flask import (
    Flask,
    render_template,
    request,
    redirect,
    url_for,
    flash,
    jsonify,
)
from flask_login import (
    LoginManager,
    login_user,
    logout_user,
    login_required,
    current_user,
)
from flask_bcrypt import Bcrypt
from app import app, db, bcrypt, login_manager
from .models import User, Product, Order, OrderItem
from sqlalchemy import func, extract
from sqlalchemy.exc import SQLAlchemyError
import calendar


@app.route("/")
def home():
    return render_template("index.html")
```

```python
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        user = User.query.filter_by(username=username).first()
        if user and user.check_password(password):
            login_user(user)
            return redirect(url_for("invoice"))
        else:
            flash("Invalid username or password", "danger")
    return render_template("login.html")


@app.route("/signup", methods=["GET", "POST"])
def signup():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        existing_user = User.query.filter_by(username=username).first()
        if existing_user is None:
            hashed_password = bcrypt.generate_password_hash(password).decode(
                "utf-8"
            )
            new_user = User(username=username, password_hash=hashed_password)
            db.session.add(new_user)
            db.session.commit()
            return redirect(url_for("login"))
        flash("Username already exists")
    return render_template("signup.html")


@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(url_for("login"))
```

```python
@app.route("/invoice", methods=["GET", "POST"])
def invoice():
    if request.method == "POST":
        # This part handles the POST request when the order is finalized
        items = request.json["items"]
        if not items:
            return (
                jsonify({"status": "error", "message": "No items provided"}),
                400,
            )
        try:
            new_order = Order(
                user_id=current_user.id,
                total_amount=0,
                gst_amount=0,
                final_amount=0,
                status="Pending",
            )
            db.session.add(new_order)
            db.session.flush()
            total_price = 0
            for item in items:
                product = Product.query.get(item["product_id"])
                if product and item["quantity"] > 0:
                    subtotal = product.price * item["quantity"]
                    order_item = OrderItem(
                        order=new_order,
                        product=product,
                        quantity=item["quantity"],
                        item_price=subtotal,
                    )
                    db.session.add(order_item)
                    total_price += subtotal
            if total_price == 0:
                db.session.rollback()
                return (
                    jsonify(
                        {
                            "status": "error",
                            "message": "Total price cannot be zero",
```

```
                    }
                ),
                400,
            )
        gst = total_price * 0.18  # Assuming 18% GST
        new_order.total_amount = total_price
        new_order.gst_amount = gst
        new_order.final_amount = total_price + gst

        db.session.commit()
        return jsonify({"status": "success", "order_id": new_order.id})
    except Exception as e:
        db.session.rollback()
        return jsonify({"status": "error", "message": str(e)}), 500
try:
    # GET request will fetch the page initially and after order completion
    products = Product.query.all()
    return render_template("invoice.html", products=products)
except SQLAlchemyError as e:
    return jsonify({"status": "error", "message": str(e)}), 500


@app.route("/history")
@login_required
def history():
    orders = (
        Order.query.filter_by(user_id=current_user.id)
        .order_by(Order.order_time.desc())
        .all()
    )
    return render_template("history.html", orders=orders)


@app.route("/analytics")
@login_required
def analytics():
    today = (
        date.today()
    )  # Ensure we're working with date only, removing time component
    start_of_week = today - timedelta(
```

```
    days=today.weekday()
)  # Monday of this week
start_of_month = today.replace(day=1)  # First day of the current month
start_of_year = today.replace(month=1, day=1)

daily_sales = (
    db.session.query(func.sum(Order.final_amount))
    .filter(func.date(Order.order_time) == today)
    .scalar()
    or 0
)
weekly_sales = (
    db.session.query(func.sum(Order.final_amount))
    .filter(func.date(Order.order_time) >= start_of_week)
    .scalar()
    or 0
)
monthly_sales = (
    db.session.query(func.sum(Order.final_amount))
    .filter(func.date(Order.order_time) >= start_of_month)
    .scalar()
    or 0
)
yearly_sales = (
    db.session.query(func.sum(Order.final_amount))
    .filter(func.date(Order.order_time) >= start_of_year)
    .scalar()
    or 0
)

# Round sales data to 2 decimal places for better presentation
daily_sales = round(daily_sales, 2)
weekly_sales = round(weekly_sales, 2)
monthly_sales = round(monthly_sales, 2)
yearly_sales = round(yearly_sales, 2)

# Popular products
popular_products = (
    db.session.query(
        Product.name, func.sum(OrderItem.quantity).label("total_sold")
```

```
    )
    .join(OrderItem, OrderItem.product_id == Product.id)
    .group_by(Product.name)
    .order_by(func.sum(OrderItem.quantity).desc())
    .limit(5)
    .all()
)

# Monthly sales for the last 12 months
monthly_sales_data = (
    db.session.query(
        extract("year", Order.order_time).label("year"),
        extract("month", Order.order_time).label("month"),
        func.sum(Order.final_amount).label("total"),
    )
    .filter(Order.order_time >= (today - timedelta(days=365)))
    .group_by(
        extract("year", Order.order_time),
        extract("month", Order.order_time),
    )
    .order_by(
        extract("year", Order.order_time),
        extract("month", Order.order_time),
    )
    .all()
)

monthly_sales_last_year = [
    (f"{calendar.month_name[month]} {year}", round(total, 2))
    for year, month, total in monthly_sales_data
]

product_sales = (
    db.session.query(
        Product.name,
        func.sum(OrderItem.quantity * Product.price).label("total_sales"),
    )
    .join(OrderItem, Product.id == OrderItem.product_id)
    .group_by(Product.name)
    .all()
```

```
    )

    return render_template(
        "analytics.html",
        daily_sales=daily_sales,
        weekly_sales=weekly_sales,
        monthly_sales=monthly_sales,
        yearly_sales=yearly_sales,
        monthly_sales_last_year=monthly_sales_last_year,
        popular_products=popular_products,
        product_sales=product_sales,
    )
```

/app/models.py

```
from app import db, app, bcrypt
import datetime
from flask_login import UserMixin


class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password_hash = db.Column(db.String(120), nullable=False)

    orders = db.relationship("Order", backref="user", lazy=True)

    def set_password(self, password):
        self.password_hash = bcrypt.generate_password_hash(password)

    def check_password(self, password):
        return bcrypt.check_password_hash(self.password_hash, password)


class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    size = db.Column(db.String(20), nullable=False)
```

```
    price = db.Column(db.Float, nullable=False)

    order_items = db.relationship("OrderItem", backref="product", lazy=True)


class Order(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"), nullable=False)
    total_amount = db.Column(db.Float)
    gst_amount = db.Column(db.Float)
    final_amount = db.Column(db.Float)
    order_time = db.Column(
        db.DateTime, default=datetime.datetime.now(datetime.UTC)
    )
    status = db.Column(db.String(20), default="Pending")

    items = db.relationship("OrderItem", backref="order", lazy=True)


class OrderItem(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    order_id = db.Column(
        db.Integer, db.ForeignKey("order.id"), nullable=False
    )
    product_id = db.Column(
        db.Integer, db.ForeignKey("product.id"), nullable=False
    )
    quantity = db.Column(db.Integer, nullable=False)
    item_price = db.Column(db.Float, nullable=False)


with app.app_context():
    db.create_all()
```

/app/auth.py

```
from app import login_manager
```

```
from .models import User


@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

/app/__init__.py:

```
import os
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_bcrypt import Bcrypt
from flask_login import LoginManager
from dotenv import load_dotenv


load_dotenv()

app = Flask(__name__)
app.config["SECRET_KEY"] = os.getenv("SECRET_KEY")
app.config["SQLALCHEMY_DATABASE_URI"] = os.getenv("SQLALCHEMY_DATABASE_URI")
db = SQLAlchemy(app)
bcrypt = Bcrypt(app)

login_manager = LoginManager()
login_manager.init_app(app)

from app import routes, models, auth
```

run.py:

```
from app import app
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

/app/templates/base.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>{% block title %}{% endblock %}</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" />
    <style>
      body {
        background-color: #f8f9fa;
        font-family: Arial, sans-serif;
      }
      .navbar {
        background-color: #007bff; /* A vibrant blue */
        padding: 10px 0;
      }
      .navbar-brand {
        font-size: 2rem; /* Larger brand font */
        color: #0062cc !important; /* Ensuring the brand text is white */
        position: absolute;
        left: 50%;
        transform: translateX(-50%);
        top: 10px;
        font-weight: bold;
        text-transform: uppercase;
      }
      .navbar-nav .nav-link.active {
        background-color: #0062cc; /* Highlighting active link */
        color: #fff;
      }
      .container,
      .card {
        margin-top: 20px;
```

```
    }
    .form-label {
      font-weight: bold;
    }
    input[type="number"] {
      width: 100%;
    }
    .row {
      margin-top: 10px;
    }
    footer {
      background-color: #007bff; /* Matching the navbar */
      color: #fff;
      text-align: center;
      padding: 10px 0;
      position: fixed; /* Fixed to the bottom of the viewport */
      bottom: 0;
      width: 100%;
    }
  </style>
</head>
<body>
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
      <a class="navbar-brand" href="{{ url_for('home') }}">Ice Cream Wala</a>
      <button
        class="navbar-toggler"
        type="button"
        data-bs-toggle="collapse"
        data-bs-target="#navbarNavAltMarkup"
        aria-controls="navbarNavAltMarkup"
        aria-expanded="false"
        aria-label="Toggle navigation"
      >
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
        <div class="navbar-nav ms-auto">
          {% if current_user.is_authenticated %}
```

```
          <a class="nav-link {% if request.endpoint == 'invoice' %}active{% endif %}" href="{{ url_for('invoice')
}}">Invoice</a>
          <a class="nav-link {% if request.endpoint == 'history' %}active{% endif %}" href="{{ url_for('history')
}}">History</a>
          <a class="nav-link {% if request.endpoint == 'analytics' %}active{% endif %}" href="{{ url_for('analytics')
}}">Analytics</a>
          <a class="nav-link" href="{{ url_for('logout') }}">Logout</a>
          {% else %}
          <a class="nav-link {% if request.endpoint == 'login' %}active{% endif %}" href="{{ url_for('login')
}}">Login</a>
          <a class="nav-link {% if request.endpoint == 'signup' %}active{% endif %}" href="{{ url_for('signup')
}}">Sign Up</a>
          {% endif %}
        </div>
      </div>
    </div>
  </nav>
  <div class="container mt-6">
    {% with messages = get_flashed_messages(with_categories=true) %}
      {% if messages %}
        {% for category, message in messages %}
          <div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">
            {{ message }}
          </div>
        {% endfor %}
      {% endif %}
    {% endwith %}
    <div class="row justify-content-center">
      {% block content %}{% endblock %}
    </div>
  </div>
  <!-- <footer>
    <p>© 2024 Ice Cream Wala. All rights reserved.</p>
  </footer> -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
 </body>
</html>
```

/app/templates/signup.html:

```
{% extends "base.html" %}
{% block title %}Sign Up{% endblock %}

{% block content %}
<div class="col-md-6">
    <div class="card">
        <div class="card-header">Sign Up</div>
        <div class="card-body">
            <form action="/signup" method="post">
                <div class="mb-3">
                    <label for="username" class="form-label">Username</label>
                    <input type="text" class="form-control" id="username" name="username" required>
                </div>
                <div class="mb-3">
                    <label for="password" class="form-label">Password</label>
                    <input type="password" class="form-control" id="password" name="password" required>
                </div>
                <button type="submit" class="btn btn-primary">Register</button>
            </form>
        </div>
    <p class="text-center mt-3">
        Already have an account? <a href="/login">Login</a>
    </p>
    </div>
</div>
{% endblock %}
```

/app/templates/login.html:

```
{% extends "base.html" %}
{% block title %}Login{% endblock %}

{% block content %}
<div class="col-md-6">
```

```
  <div class="card">
    <div class="card-header">Login</div>
    <div class="card-body">
      <form action="/login" method="post">
        <div class="mb-3">
          <label for="username" class="form-label">Username</label>
          <input type="text" class="form-control" id="username" name="username" required>
        </div>
        <div class="mb-3">
          <label for="password" class="form-label">Password</label>
          <input type="password" class="form-control" id="password" name="password" required>
        </div>
        <button type="submit" class="btn btn-primary">Login</button>
      </form>
    </div>
      <p class="text-center mt-3">
    Don't have an account? <a href="/signup">Sign up</a>
  </p>
  </div>
</div>
{% endblock %}
```

/app/templates/invoice.html:

```
{% extends "base.html" %}
{% block title %}Create Invoice{% endblock %}

{% block content %}
<div class="container">
  <h2>Create New Order</h2>
  <button id="newOrderBtn" class="btn btn-primary mb-3">Create New Order</button>
  <div id="orderForm" style="display:none;">
    <form id="addItemForm">
      <div class="mb-3">
        <label for="productSelect" class="form-label">Product</label>
        <select id="productSelect" class="form-select" required onchange="updateSizes()">
          <option value="">Select Product</option>
          {% for product in products %}
```

```
            {% if not loop.first %}
            {% if products[loop.index - 2].name != product.name %}
            <option value="{{ product.name }}">{{ product.name }}</option>
            {% endif %}
            {% else %}
            <option value="{{ product.name }}">{{ product.name }}</option>
            {% endif %}
            {% endfor %}
        </select>
    </div>
    <div class="mb-3">
        <label for="sizeSelect" class="form-label">Size</label>
        <select id="sizeSelect" class="form-select" required>
            <!-- Sizes will be populated based on product selection -->
        </select>
    </div>
    <div class="mb-3">
        <label for="quantity" class="form-label">Quantity</label>
        <input type="number" id="quantity" class="form-control" required min="1">
    </div>
    <button type="button" id="addItemBtn" class="btn btn-secondary">Add Item</button>
</form>
<h4>Order Details:</h4>
<table class="table">
    <thead>
        <tr>
            <th>Product</th>
            <th>Size</th>
            <th>Price</th>
            <th>Quantity</th>
            <th>Subtotal</th>
        </tr>
    </thead>
    <tbody id="orderItems">
        <!-- Items will be added here dynamically -->
    </tbody>
</table>
<div id="paymentMethod">
    <label class="form-label">Payment Method:</label>
    <div class="form-check form-check-inline">
```

```
            <input class="form-check-input" type="radio" name="paymentMethod" id="cashPayment" value="cash"
checked>
            <label class="form-check-label" for="cashPayment">Cash</label>
          </div>
          <div class="form-check form-check-inline">
            <input class="form-check-input" type="radio" name="paymentMethod" id="cardPayment"
value="card">
            <label class="form-check-label" for="cardPayment">Card</label>
          </div>
        </div>
        <div id="invoiceSummary">
          <p>GST (18%): <span id="gstAmount">$0</span></p>
          <p>Grand Total: <span id="grandTotal">$0</span></p>
        </div>
        <button type="button" id="completeOrderBtn" class="btn btn-success">Complete Order</button>
    </div>
</div>

<script>
function updateSizes() {
    var productSelect = document.getElementById('productSelect');
    var sizeSelect = document.getElementById('sizeSelect');
    var selectedProduct = productSelect.value;
    sizeSelect.innerHTML = ''; // Clear previous options

    {% for product in products %}
    if ("{{ product.name }}" == selectedProduct) {
        var option = document.createElement('option');
        option.value = "{{ product.id }}";
        option.textContent = "{{ product.size }} - ${{ product.price }}";
        option.setAttribute('data-price', "{{ product.price }}");
        sizeSelect.appendChild(option);
    }
    {% endfor %}
}

document.getElementById('newOrderBtn').addEventListener('click', function() {
    document.getElementById('orderForm').style.display = 'block';
    document.getElementById('orderItems').innerHTML = ''; // Clear previous items
    document.getElementById('gstAmount').textContent = '$0';
```

```
    document.getElementById('grandTotal').textContent = '$0';
});

document.getElementById('addItemBtn').addEventListener('click', function() {
    var sizeSelect = document.getElementById('sizeSelect');
    var quantityInput = document.getElementById('quantity');
    var quantity = parseInt(quantityInput.value);
    if (quantity < 1) {
        alert('Quantity must be at least 1');
        return;
    }
    var selectedOption = sizeSelect.options[sizeSelect.selectedIndex];
    var product =
document.getElementById('productSelect').options[document.getElementById('productSelect').selectedIndex].tex
t;
    var size = selectedOption.text.split(' - ')[0];
    var price = selectedOption.getAttribute('data-price');
    var productId = selectedOption.value;
    var subtotal = parseFloat(price) * parseInt(quantity);

    var newRow = `<tr data-product-id="${productId}">
        <td>${product}</td>
        <td>${size}</td>
        <td>$${price}</td>
        <td>${quantity}</td>
        <td>$${subtotal.toFixed(2)}</td>
    </tr>`;

    document.getElementById('orderItems').insertAdjacentHTML('beforeend', newRow);

    // Reset the form inputs
    quantityInput.value = '';
    updateInvoiceSummary();
});

function updateInvoiceSummary() {
    var total = 0;
    document.querySelectorAll('#orderItems tr').forEach(row => {
        total += parseFloat(row.cells[4].textContent.substring(1)); // remove the dollar sign and parse
    });
```

```
    var gst = total * 0.18; // 18% GST
    var grandTotal = total + gst;

    document.getElementById('gstAmount').textContent = `${gst.toFixed(2)}`;
    document.getElementById('grandTotal').textContent = `${grandTotal.toFixed(2)}`;
}

document.getElementById('completeOrderBtn').addEventListener('click', function() {
    var items = [];
    var rows = document.getElementById('orderItems').querySelectorAll('tr');
    rows.forEach(row => {
        items.push({
            product_id: row.getAttribute('data-product-id'),
            quantity: parseInt(row.cells[3].textContent)
        });
    });

    fetch('/invoice', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({items: items})
    }).then(response => response.json())
      .then(data => {
        if(data.status === 'success') {
            alert('Order completed successfully!');
            document.getElementById('orderForm').style.display = 'none';
        } else {
            alert('Error completing order.');
        }
    });
});
</script>
{% endblock %}
```

/app/templates/index.html:

```
{% extends "base.html" %}

{% block content %}
<style>
    .centered-content {
        display: flex;
        justify-content: center;
        align-items: center;
        height: 80vh; /* Adjust height as needed */
        text-align: center;
    }

    .background {
        background-image: url('icecream.jpg');
        background-size: cover;
        background-position: center;
    }

    .animated-element {
        animation: fadeIn 2s;
    }

    @keyframes fadeIn {
        from { opacity: 0; }
        to { opacity: 1; }
    }
</style>

<div class="background">
    <div class="centered-content">
        <div class="animated-element">
            <h1 class="display-4">Welcome to Ice Cream Wala</h1>
            <p class="lead">Manage your ice cream orders and sales effortlessly in one place.</p>
            <hr class="my-4">
            <a class="btn btn-primary btn-lg" href="/login" role="button">Get Started</a>
        </div>
    </div>
</div>
{% endblock %}
```

/app/templates/history.html:

```
{% extends "base.html" %}
{% block title %}Order History{% endblock %}

{% block content %}
<div class="container mt-4">
    <h2>Order History</h2>
    {% for order in orders %}
    <div class="card mb-3">
        <div class="card-header">
            Order Placed on: {{ order.order_time.strftime('%Y-%m-%d %H:%M:%S') }}
        </div>
        <div class="card-body">
            <h5 class="card-title">Order Details</h5>
            <table class="table">
                <thead>
                    <tr>
                        <th>Product</th>
                        <th>Size</th>
                        <th>Price</th>
                        <th>Quantity</th>
                        <th>Subtotal</th>
                    </tr>
                </thead>
                <tbody>
                    {% for item in order.items %}
                    <tr>
                        <td>{{ item.product.name }}</td>
                        <td>{{ item.product.size }}</td>
                        <td>${{ item.product.price }}</td>
                        <td>{{ item.quantity }}</td>
                        <td>${{ item.item_price }}</td>
                    </tr>
                    {% endfor %}
                </tbody>
            </table>
            <p class="card-text"><strong>Total Price:</strong> ${{ order.total_amount }}</p>
```

```
            <p class="card-text"><strong>GST:</strong> ${{ order.gst_amount }}</p>
            <p class="card-text"><strong>Grand Total:</strong> ${{ order.final_amount }}</p>
        </div>
    </div>
    {% else %}
    <p>No orders found.</p>
    {% endfor %}
</div>
{% endblock %}
```

/app/templates/analytics.html:

```
{% extends "base.html" %}
{% block title %}Analytics{% endblock %}

{% block content %}
<div class="container mt-4">
    <h1 class="text-center mb-4">Sales Analytics Dashboard</h1>

    <div class="row text-center mb-4">
        <div class="col-md-3">
            <div class="card">
                <div class="card-body">
                    <h5 class="card-title">Daily Sales</h5>
                    <p class="card-text">${{ daily_sales }}</p>
                </div>
            </div>
        </div>
        <div class="col-md-3">
            <div class="card">
                <div class="card-body">
                    <h5 class="card-title">Weekly Sales</h5>
                    <p class="card-text">${{ weekly_sales }}</p>
                </div>
            </div>
        </div>
```

```
    <div class="col-md-3">
        <div class="card">
            <div class="card-body">
                <h5 class="card-title">Monthly Sales</h5>
                <p class="card-text">${{ monthly_sales }}</p>
            </div>
        </div>
    </div>
    <div class="col-md-3">
        <div class="card">
            <div class="card-body">
                <h5 class="card-title">Yearly Sales</h5>
                <p class="card-text">${{ yearly_sales }}</p>
            </div>
        </div>
    </div>
</div>

<div class="row">
    <div class="col-md-6">
        <h3>Top 5 Popular Products</h3>
        <ul class="list-group">
            {% for product, total_sold in popular_products %}
            <li class="list-group-item d-flex justify-content-between align-items-center">
                {{ product }}
                <span class="badge bg-primary rounded-pill">{{ total_sold }}</span>
            </li>
            {% endfor %}
        </ul>
    </div>
    <div class="col-md-6">
        <h3>Monthly Sales</h3>
        <table class="table">
            <thead>
                <tr>
                    <th>Month</th>
                    <th>Total Sales</th>
                </tr>
            </thead>
            <tbody>
```

```
                {% for month, total in monthly_sales_last_year %}
                <tr>
                    <td>{{ month }}</td>
                    <td>${{ total }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
</div>

<div class="mt-4 text-center">
    <h3>Sales Distribution by Product</h3>
    <div id="productSalesPieChart" style="display: inline-block; width: 100%; max-width: 550px; height:
400px;"></div>
</div>
</div>

<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
    // Load google charts
    google.charts.load('current', {'packages':['corechart']});
    google.charts.setOnLoadCallback(drawProductSalesChart);

    // Draw the pie chart for product sales
    function drawProductSalesChart() {
        var data = google.visualization.arrayToDataTable([
            ['Product', 'Sales'],
            {% for product, total_sales in product_sales %}
            ['{{ product }}', {{ total_sales }}],
            {% endfor %}
        ]);

        var options = {
            title: 'Product Sales Distribution',
            width: '100%',
            height: '100%',
            pieHole: 0.4,  // Optional: for donut style
        };
```

```
        var chart = new google.visualization.PieChart(document.getElementById('productSalesPieChart'));
        chart.draw(data, options);
    }
</script>
{% endblock %}
```