

ITCS 5102 - SURVEY OF
PROGRAMMING LANGUAGES
TERM PROJECT FINAL REPORT

TEAM PROFILE:

Team Members	Student ID	UNCC Email ID
Nikhita Bharat Upadhye	801337745	nupadhye@uncc.edu
Rakesh Moramreddy	801337430	rmoramre@uncc.edu
Srinivasulu Padigay	801330017	spadigay@uncc.edu
Vachanshree Mugappa Chabbi	801306456	vchabbi@uncc.edu

Introduction - Motivation:

In the dynamic landscape of modern business, efficient sales management stands as a cornerstone for sustainable growth and success. Recognizing this pivotal role, the Sales Tracker project emerges as a comprehensive solution designed to streamline sales processes and optimize business performance.

The motivation behind this project stems from the imperative need within businesses for a robust sales management application that not only enhances operational efficiency but also empowers users with insightful data-driven decision-making capabilities. In today's competitive market environment, businesses are increasingly reliant on sophisticated tools to manage sales activities, nurture customer relationships, and drive revenue growth.

The Sales Tracker project addresses these fundamental needs by combining cutting-edge technologies, including a React frontend and a GoLang backend, to deliver a seamless and intuitive user experience. By offering a centralized platform for tracking sales activities and managing customer interactions, this application aims to revolutionize the way businesses approach sales management.

Through this report, we aim to provide a comprehensive overview of the Sales Tracker project, delving into its architecture, functionalities, and potential impact on business operations. Additionally, we will explore the underlying motivations driving the development of this solution and explain its significance in the context of contemporary business challenges.

Ultimately, this report serves as a testament to our commitment to innovation and excellence in addressing the evolving needs of businesses worldwide. By harnessing the power of technology and strategic insights, the Sales Tracker project endeavors to empower organizations to thrive in an increasingly competitive marketplace.

Language Specifications:

1. Paradigm: Go is a statically typed, compiled language with a focus on simplicity and concurrency, blending procedural and object-oriented programming.
2. Historical Account: Created by Google in 2007, Go aimed to address challenges in large-scale software development, drawing influences from languages like C and Pascal.
3. Elements of the Language:
 - Reserved words: Small set including var, func, if, etc.
 - Primitive data types: Basic types such as int, string, etc.
 - Structured types: Arrays, slices, maps, structs, and pointers.
4. Syntax: C-like syntax prioritizing readability with optional semicolons and curly braces to denote code blocks.
5. Basic Control Abstractions: Offers standard control structures like loops, conditional statements, and switch-case constructs.
6. Abstraction Handling:
 - Functions: First-class citizens supporting assignment, passing, and return.
 - Packages: Encourages modular programming for encapsulation and reusability.
 - Interfaces: Define behavior for polymorphism and decoupling.
7. Writability, Readability, Reliability:
 - Writability: Promotes concise, expressive code with built-in features like concurrency.
 - Readability: Minimalistic syntax and idiomatic conventions enhance code clarity.
 - Reliability: Strong type system, explicit error handling, and concurrency primitives bolster reliability.
8. Strengths and Weaknesses:
 - Strengths: Concurrency, simplicity, fast compilation, built-in testing, strong standard library.
 - Weaknesses: Lack of generics, limited tooling, and slower execution speed compared to lower-level languages in some cases.
9. Program Overview: Demonstrates building a REST API for managing sales records, showcasing Go's simplicity, concurrency, and readability.
10. Sample Run: Compiling and running the program using go run or go build commands, then accessing defined endpoints via an HTTP client like cURL or a web browser.

The paradigm of the language:

Go is a statically typed, compiled language designed for simplicity and efficiency. It follows the imperative and procedural programming paradigms with support for concurrent programming through goroutines and channels.

Some historical account of the evolution of the language and its

Antecedents:

Go was developed by Google engineers Rob Pike, Ken Thompson, and Robert Griesemer in 2007 and was publicly released in 2009. It was designed as a response to the shortcomings of other languages in Google's internal projects, aiming for simplicity, efficiency, and ease of use.

The elements of the language: reserved words, primitive data types, structured types:

Reserved words: Go has a set of reserved keywords such as func, var, if, else, for, switch, case, select, etc.

Primitive data types: Go has primitive types such as int, float64, string, bool, etc.

Structured types: Go supports structured types like struct, arrays, slices, maps, channels, pointers, etc.

A description (in some form) of the syntax of the language:

Go has a C-like syntax with a clear and concise structure. It emphasizes readability and simplicity.

The basic control abstractions of the language (loops, conditional controls, etc.):

Go supports basic control structures like loops (for), conditional controls (if, else, switch), and defer statements.

How the language handles abstraction (including functions, procedures, objects, modules, etc.):

Go supports various levels of abstraction including functions, methods (via structs), interfaces, and packages/modules. It emphasizes composition over inheritance.

An evaluation of the language's writability, readability, and reliability:

Go is highly writable due to its concise syntax and built-in support for features like concurrency. Its readability is excellent due to its simplicity and strong conventions. Go's reliability is high due to its static typing, strong standard library, and memory safety.

The major strengths and weaknesses of your language:

Strengths: Concurrency support, simplicity, strong standard library, fast compilation, static typing, garbage collection.

Weaknesses: Lack of generics (until recent versions), smaller ecosystem compared to some other languages, less expressive error handling.

An overview of the programs that you included and a discussion of : what language features they highlight and how the language made the programs easy/hard to implement:

Programs in Go often highlight its simplicity, concurrency features, and ease of use for network programming, web development, and system-level programming.

Sample run of a program in the language and/or platform:

Below is a simple program in Go that prints "Hello, World!"-

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

To run this program, save it in a file named hello.go and execute `go run hello.go` in your terminal. It should print Hello, World!

Problem definition(Include use cases):

Problem Definition:

The problem that the Sales Tracker project seeks to address revolves around the need for an efficient and user-friendly sales management application. Businesses often struggle with disparate systems for tracking sales activities, managing customer relationships, and analyzing sales data. This fragmented approach leads to inefficiencies, data silos, and missed opportunities for revenue growth.

Test Cases:

1. Scenario: Shopkeeper Tracks Daily Sales

- Input: Shopkeeper enters sales data for the day into different categories (e.g., jewelry, stationary, clothing etc).
- Action: Enter sales quantities and prices for each category.
- Expected Output: Sales data for the day is recorded and categorized accordingly in the system.

2. Scenario: Adding Sales Record for a Commodity

- Input: Shopkeeper adds a new sales record for a commodity, specifying details such as item name, price, and payment method (cash or card).
- Action: Enter details of the commodity sold and select payment method.
- Expected Output: New sales record is added to the system with specified details and payment method.

3. Scenario: Modifying and Updating Sales Record

- Input: Shopkeeper identifies an error or change in a sales record and needs to modify it.
- Action: Locate the sales record to be modified, update the relevant information (e.g., price, payment method), and save changes.
- Expected Output: Sales record is successfully updated with the new information.

4. Scenario: Automatically Displaying Total Sales of the Day by Payment Method

- Input: Sales records for the day categorized by payment method (cash or card).
- Action: Upon entering or modifying sales records, the system automatically updates and displays the total sales amount for the day, separately for cash transactions and card transactions.
- Expected Output: The total sales amount for the day is dynamically calculated and shown in real-time, with separate values for cash transactions and card transactions, providing immediate insights into the distribution of payment methods used by customers.

5. Scenario: Exporting Daily Sales as PDF

- Input: Shopkeeper wants to export the daily sales data as a PDF document for record-keeping or sharing with the manager.
- Action: Select the option to export daily sales, choose PDF format, and it is downloaded.
- Expected Output: Daily sales data is exported as a PDF document and saved in the downloads.

Proposed method if any:

The software system described comprises a backend in Go and a frontend using React.js, structured around sales data management. The system allows users to add, update, and delete sales records, and also view daily totals, and generate PDF reports of sales.

Intuition - why should it be better than the state of the art?:

The algorithm begins by focusing on designing a user-centric interface that is intuitive and easy to navigate, optimizing data input through smart forms and auto-fill functionalities to reduce manual effort. Real-time visualization of sales data provides immediate insights into sales trends and performance, with interactive charts dynamically updating as new data is entered. Automated analysis capabilities detect patterns and anomalies in sales data, while intelligent notification systems alert users of important events or actions required. Adaptive recommendations based on historical sales data and trends suggest personalized strategies for optimizing sales and maximizing revenue. Seamless integration with existing systems ensures compatibility and interoperability, minimizing disruption to workflow processes and enhancing overall efficiency.

Description of its algorithms:

1. Sales Data Management:

- **Add/Update Sales:** Uses HTTP POST/PUT requests to send data to the server where it is processed and stored in a database.
- **Delete Sales:** Uses an HTTP DELETE request to remove sales records from the database.
- **Fetch Sales:** Periodically fetches all sales records from the server using an HTTP GET request.

2. Total Calculations:

- **Daily Totals:** Accumulates totals from sales entries stored in state, updated anytime a sales operation is performed.

3. PDF Generation:

- **Capture UI Elements:** Uses html2canvas to capture rendered HTML as an image.
- **PDF Creation:** Converts the captured image to a PDF using jsPDF.

Experiments:

1. Performance Benchmarking:

- Test response times for various operations (add, update, delete) under different network conditions.
- Measure server load during simultaneous operations from multiple users.

2. Usability Testing:

- Conduct user tests to assess the intuitiveness of the UI and gather feedback on potential improvements.
- Evaluate the effectiveness of the new features like real-time updates and advanced reporting.

Description of your test bed; list of questions your experiments are designed to answer:

- **Servers and Clients:**
 - Multiple client machines to simulate different user environments.
 - A dedicated server hosting the backend and serving the frontend.
- **Tools and Technologies:**
 - Network simulation tools to mimic various internet conditions.
 - Security testing tools for performing authentication tests and vulnerability scans.
- **Metrics Collected:**
 - Response times for each type of request.
 - CPU and memory usage on the server during tests.
 - User satisfaction ratings from usability testing.

List of Questions Designed to Answer Through Experiments

1. How does the system perform under high load conditions?
2. Is the system's user interface intuitive and easy to use for typical tasks?
3. Are there any security vulnerabilities in the current implementation?
4. How effective are the real-time updates in improving user experience?
5. Does the advanced reporting functionality meet the users' needs for data analysis?

Details of the experiments; observations:

Adding a Sale

Frontend (React.js)

- **Interface:** The SaleForm component provides a form where users can enter details about a sale, such as item name, price, date, category, and payment mode.
- **Context and State Management:** The SalesContext manages the state using React's useContext hook. It holds the current form data and provides a method to update this data (setFormData).

- **Submission:** When the "Add Sale" button is clicked, if the form is not in edit mode (determined by the editing flag in formData), the addOrUpdateSale function is triggered.
- **API Interaction:** This function sends a POST request to the backend with the sale data. After the request is successful, it fetches the updated sales data and today's totals to refresh the local state.

Backend (Go)

- **API Endpoint:** A typical handler for adding a sale would receive the POST request, parse the sale data, and perform validation.
- **Database Interaction:** If the data is valid, it is inserted into the database.
- **Response:** The server then responds with the success status and possibly the details of the added sale.

Updating a Sale

Frontend

- **Start Edit:** The SaleList component provides an edit button next to each sale. When clicked, it triggers the startEdit method in SalesContext, which populates the form data with the sale's current details and sets the editing flag to true.
- **Form Reuse:** The SaleForm is reused for updates. Since the form is pre-filled with the sale details, users can modify them.
- **Submission:** Submitting the form while in edit mode triggers the addOrUpdateSale function, which this time sends a PUT request to update the existing sale based on its ID.

Backend

- **API Endpoint:** The PUT endpoint captures the sale ID, validates the incoming data, and updates the corresponding record in the database.
- **Response:** The server responds with the updated record or an error message if applicable.

Deleting a Sale

Frontend

- **Delete Option:** In the SaleList component, a delete icon appears next to each sale item.
- **Deletion Functionality:** Clicking the delete icon triggers the deleteSale function in SalesContext.
- **API Interaction:** This function sends a DELETE request to the backend with the ID of the sale to be deleted.

Backend

- **API Endpoint:** The DELETE endpoint receives the sale ID, performs necessary authorization checks, and deletes the sale from the database.
- **Response:** The server sends a success or error response back to the client.

Observations

- **Efficiency:** The frontend uses React for efficient updates to the UI without reloading the page, utilizing the context API for state management across components.
- **Security:** The current implementation does not include user authentication, which is a potential security risk if sensitive data is being handled.

- **Scalability:** The backend, typically written in Go, is well-suited for handling high concurrency, making the system scalable.
- **Maintainability:** The separation of concerns between the frontend and backend allows for easier maintenance and updates to the system.

Screenshot of codes:

Backend Go lang code:

main.go

```

package main

import (
    "database/sql"
    "github.com/gin-contrib/cors"
    "github.com/gin-gonic/gin"
    "github.com/mattn/go-sqlite3" // Use underscore if it's only for the side effect of registering the driver
    "log"
    "net/http"
    "strconv"
    "time"
)

type Sale struct {
    ID          int      `json:"id"`
    ItemName    string   `json:"itemName"`
    Price       float64  `json:"price"`
    Date        string   `json:"date"`
    PaymentMode string   `json:"paymentMode"`
    Category    string   `json:"category" // New field for category`
}

var db *sql.DB

func main() {
    var err error
    db, err = sql.Open("sqlite3", "./sales.db")
    if err != nil {
        log.Fatalf(err)
    }
    defer db.Close()

    createTable()

    r := gin.Default()
    r.Use(cors.Default())
    r.GET("/sales", getSales)
    r.POST("/sales", postSale)
    r.PUT("/sales/:id", updateSale)
    r.DELETE("/sales/:id", deleteSale)
    r.GET("/today/sales", getTodaySales) // This should be inside the main function
    r.Run(":8080")
}

func createTable() {
    createTableSQL := `
CREATE TABLE IF NOT EXISTS sales (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    itemName TEXT NOT NULL,
    price REAL NOT NULL,
    date TEXT NOT NULL,
    paymentMode TEXT NOT NULL,
    category TEXT NOT NULL
);`
    err := db.Exec(createTableSQL)
    if err != nil {
        log.Fatalf(err)
    }
}

func getSales(c *gin.Context) {
    rows, err := db.Query("SELECT id, itemName, price, date, paymentMode, category FROM sales")
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    defer rows.Close()

    var sales []Sale
    for rows.Next() {
        var sale Sale
        if err := rows.Scan(&sale.ID, &sale.ItemName, &sale.Price, &sale.Date, &sale.PaymentMode, &sale.Category); err != nil {
            c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
            return
        }
        sales = append(sales, sale)
    }
    c.JSON(http.StatusOK, sales)
}

func postSale(c *gin.Context) {
    var newSale Sale
    if err := c.ShouldBind(&newSale); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }
    stmt, err := db.Prepare("INSERT INTO sales(itemName, price, date, paymentMode, category) VALUES(?, ?, ?, ?, ?)")
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    defer stmt.Close()
    _, err = stmt.Exec(newSale.ItemName, newSale.Price, newSale.Date, newSale.PaymentMode, newSale.Category)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    c.JSON(http.StatusOK, newSale)
}

func updateSale(c *gin.Context) {
    id, err := strconv.Atoi(c.Param("id"))
    if err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid ID"})
        return
    }
    var updatedSale Sale
    if err := c.ShouldBind(&updatedSale); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }
    stmt, err := db.Prepare("UPDATE sales SET itemName=?, price=?, date=?, paymentMode=?, category=? WHERE id=?")
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    defer stmt.Close()
    _, err = stmt.Exec(updatedSale.ItemName, updatedSale.Price, updatedSale.Date, updatedSale.PaymentMode, updatedSale.Category, id)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    c.JSON(http.StatusOK, updatedSale)
}

func deleteSale(c *gin.Context) {
    id, err := strconv.Atoi(c.Param("id"))
    if err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid ID"})
        return
    }
    stmt, err := db.Prepare("DELETE FROM sales WHERE id=?")
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    defer stmt.Close()
    _, err = stmt.Exec(id)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    c.Status(http.StatusOK)
}

func getTodaySales(c *gin.Context) {
    today := (time.Now()).Format("2006-01-02") // Format today's date as YYYY-MM-DD
    query := `
SELECT paymentMode, SUM(price) as total
FROM sales
WHERE date = ?
GROUP BY paymentMode`
    rows, err := db.Query(query, today)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to query sales"})
        return
    }
    defer rows.Close()

    totals := make(map[string]float64)
    for rows.Next() {
        var paymentMode string
        var total float64
        err := rows.Scan(&paymentMode, &total)
        if err != nil {
            c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to read sales totals"})
            return
        }
        totals[paymentMode] = total
    }
    c.JSON(http.StatusOK, totals)
}

```

Front end:

SalesFrom.js

```
import React, { useContext } from 'react';
import { TextField, Button, FormControl, InputLabel, Select, MenuItem, Box } from '@mui/material';
import { SalesContext } from '../SalesContext';

function SaleForm() {
  const { formData, setFormData, addOrUpdateSale } = useContext(SalesContext);
  const handleFormChange = (event) => {
    setFormData({ ...formData, [event.target.name]: event.target.value });
  };

  return (
    <Box sx={{ '& .MuiTextField-root': { m: 1, width: '25ch' } }}>
      <TextField
        label="Item Name"
        variant="outlined"
        name="itemName"
        value={formData.itemName}
        onChange={handleFormChange}
      />
      <TextField
        label="Price"
        variant="outlined"
        name="price"
        type="number"
        value={formData.price}
        onChange={handleFormChange}
      />
      <TextField
        label="Date"
        type="date"
        variant="outlined"
        name="date"
        InputLabelProps={{ shrink: true }}
        value={formData.date}
        onChange={handleFormChange}
      />
      <FormControl fullWidth sx={{ m: 1 }}>
        <InputLabel>Category</InputLabel>
        <Select
          name="category"
          value={formData.category}
          label="Category"
          onChange={handleFormChange}
        >
          <MenuItem value="Clothes">Clothes</MenuItem>
          <MenuItem value="Jewelry">Jewelry</MenuItem>
          <MenuItem value="Bag">Bag</MenuItem>
          <MenuItem value="Stationary">Stationary</MenuItem>
          { /* Add more categories as needed */ }
        </Select>
      </FormControl>
      <FormControl fullWidth sx={{ m: 1 }}>
        <InputLabel>Payment Mode</InputLabel>
        <Select
          name="paymentMode"
          value={formData.paymentMode}
          label="Payment Mode"
          onChange={handleFormChange}
        >
          <MenuItem value="cash">Cash</MenuItem>
          <MenuItem value="card">Card</MenuItem>
        </Select>
      </FormControl>
      <Button variant="contained" color="primary" onClick={addOrUpdateSale}>
        {formData.editing ? 'Update Sale' : 'Add Sale'}
      </Button>
    </Box>
  );
}

export default SaleForm;
```

SalesList.js

```

import React, { useContext } from 'react';
import { List, ListItem, ListItemText, IconButton, Typography, Divider, Paper } from '@mui/material';
import EditIcon from '@mui/icons-material/Edit';
import DeleteIcon from '@mui/icons-material/Delete';
import Button from '@mui/material/Button';
import { SalesContext } from './SalesContext';
import './SaleList.css';
import { jsPDF } from 'jspdf';
import html2canvas from 'html2canvas';

function SaleList() {
  const { sales, startEdit, deleteSale } = useContext(SalesContext);

  if (!sales || sales.length === 0) return <Typography>No sales data available.</Typography>;

  const salesWithTotals = sales.reduce((acc, sale) => {
    const date = sale.date;
    if (!acc[date]) {
      acc[date] = {
        sales: [],
        totals: { cash: 0, card: 0 }
      };
    }
    acc[date].sales.push(sale);
    if (sale.paymentMode === 'cash') {
      acc[date].totals.cash += parseFloat(sale.price);
    } else if (sale.paymentMode === 'card') {
      acc[date].totals.card += parseFloat(sale.price);
    }
    return acc;
  }, {});

  const exportSalesToPDF = async (date, sales, totals) => {
    // Temporarily add a class to hide elements you don't want to export
    const exportButton = document.querySelector(`#export-btn-${date}`);
    exportButton.classList.add('print-hide');

    const input = document.getElementById(`sales-${date}`); // The ID of the div you want to export
    const canvas = await html2canvas(input);
    const data = canvas.toDataURL('image/png');

    // Remove the class so the button shows again after capturing for PDF
    exportButton.classList.remove('print-hide');

    const pdf = new jsPDF();
    const imgProps = pdf.getImageProperties(data);
    const pdfWidth = pdf.internal.pageSize.getWidth();
    const pdfHeight = (imgProps.height * pdfWidth) / imgProps.width;
    pdf.addImage(data, 'PNG', 0, 0, pdfWidth, pdfHeight);
    pdf.save(`sales-${date}.pdf`);
  };

  const sortedDates = Object.keys(salesWithTotals).sort((a, b) => new Date(b) - new Date(a));

  return (
    <List sx={{ width: '100%', mt: 4 }}>
      {Object.keys(salesWithTotals).sort((a, b) => new Date(b) - new Date(a)).map(date => {
        const { sales, totals } = salesWithTotals[date];
        return (
          <React.Fragment key={date}>
            <Paper elevation={2} sx={{ my: 2, p: 2 }} id={`sales-${date}`}>
              <Typography variant="h6" color="primary" sx={{ mb: 2 }}>
                Sales on {date} | Cash Sales: ${totals.cash.toFixed(2)} | Card Sales:
                ${totals.card.toFixed(2)}
              </Typography>
              <Button
                id={`export-btn-${date}`}
                variant="contained"
                onClick={() => exportSalesToPDF(date, sales, totals)}
              >
                Export to PDF
              </Button>
            </React.Fragment>
          <Divider sx={{ mb: 2 }} />
          {sales.map((sale, index) => (
            <ListItem
              key={index}
              secondaryAction={
                <>
                  <IconButton onClick={() => startEdit(sale)}><EditIcon /></IconButton>
                  <IconButton onClick={() => deleteSale(sale.id)}><DeleteIcon />
                </>
              }
            >
              <ListItemText
                primary={` ${sale.itemName} - ${sale.price.toFixed(2)} `}
                secondary={`Category: ${sale.category} | Date: ${sale.date} | Payment Mode:
                ${sale.paymentMode}`}
              />
            </ListItem>
          ))}
        </Paper>
      )}
    </List>
  );
}

export default SaleList;

```

SalesContext.js

```
import React, { createContext, useState, useEffect } from 'react';
import axios from 'axios';

export const SalesContext = createContext();

export const SalesProvider = ({ children }) => {
  const [sales, setSales] = useState([]);
  const [formData, setFormData] = useState({
    itemName: '',
    price: '',
    date: '',
    paymentMode: 'cash',
    category: '',
    editing: false,
    id: null
  });
  const [todayTotals, setTodayTotals] = useState({ cash: 0, card: 0 });

  useEffect(() => {
    fetchSales();
    fetchTodayTotals();
  }, []);

  const fetchSales = async () => {
    try {
      const response = await axios.get('http://localhost:8080/sales');
      setSales(response.data);
    } catch (error) {
      console.error('Failed to fetch sales', error);
    }
  };

  const addOrUpdateSale = async () => {
    const submissionData = {
      ...formData,
      price: parseFloat(formData.price) || 0
    };
    const method = formData.editing ? 'put' : 'post';
    const url = formData.editing ? `http://localhost:8080/sales/${formData.id}` : 'http://localhost:8080/sales';
    try {
      await axios[method](url, submissionData);
      await fetchSales();
      await fetchTodayTotals();
      setFormData({
        itemName: '',
        price: '',
        date: '',
        paymentMode: 'cash',
        category: '',
        editing: false,
        id: null
      });
    } catch (error) {
      console.error('Failed to add/update sale', error);
    }
  };

  const deleteSale = async (id) => {
    try {
      await axios.delete(`http://localhost:8080/sales/${id}`);
      await fetchSales();
      await fetchTodayTotals();
    } catch (error) {
      console.error('Failed to delete sale', error);
    }
  };

  const startEdit = (sale) => {
    setFormData({
      ...sale,
      editing: true
    });
  };

  const fetchTodayTotals = async () => {
    try {
      const response = await axios.get('http://localhost:8080/today-sales');
      setTodayTotals(response.data);
    } catch (error) {
      console.error('Failed to fetch today's totals', error);
    }
  };

  return (
    <SalesContext.Provider value={{
      sales,
      formData,
      setFormData,
      addOrUpdateSale,
      deleteSale,
      startEdit, // Include startEdit here
      todayTotals
    }}>
      {children}
    </SalesContext.Provider>
  );
};
```

Totals.js

```

import React, { useContext } from 'react';
import { SalesContext } from '../SalesContext';
import { Typography } from '@mui/material';

function TodayTotals() {
  const { todayTotals } = useContext(SalesContext); // Use the updated context
  return (
    <div>
      <Typography variant="h6">Today's Totals</Typography>
      <Typography>Cash: ${todayTotals.cash.toFixed(2)}</Typography>
      <Typography>Card: ${todayTotals.card.toFixed(2)}</Typography>
    </div>
  );
}

export default TodayTotals;

```

Screenshot of app:

Adding a cash transaction:

Sales Tracker

Item Name milk	Price 2.99	Date 04/21/2024
Category <div> Clothes Jewelry Bag Stationary </div>		

Sales Tracker

Item Name

Price

Date

mm/dd/yyyy

Category

Payment Mode

Cash

ADD SALE

Sales on 2024-04-21 | Cash Sales: \$2.99 | Card Sales: \$0.00

EXPORT TO PDF

milk - \$2.99

Category: Stationary | Date: 2024-04-21 | Payment Mode: cash

Adding a card transaction:

Sales Tracker

Item Name

chocolate

Price

10.99

Date

02/21/2024

Category

Bag

Payment Mode

Card

ADD SALE

Sales on 2024-04-21 | Cash Sales: \$2.99 | Card Sales: \$0.00EXPORT TO PDF

milk - \$2.99

Category: Stationary | Date: 2024-04-21 | Payment Mode: cash

Sales Tracker

Item Name

Price

Date

mm/dd/yyyy

Category

Payment Mode

Cash

ADD SALE

Sales on 2024-04-21 | Cash Sales: \$2.99 | Card Sales: \$10.99EXPORT TO PDF

milk - \$2.99

Category: Stationary | Date: 2024-04-21 | Payment Mode: cash

chocolate - \$10.99

Category: Bag | Date: 2024-04-21 | Payment Mode: card

Updating cash transaction:

Sales Tracker

Item Name

milk

Price

6.99

Date

04/21/2024

Category

Stationary

Payment Mode

Cash

UPDATE SALE

Sales on 2024-04-21 | Cash Sales: \$2.99 | Card Sales: \$10.99EXPORT TO PDF

milk - \$2.99 Category: Stationary Date: 2024-04-21 Payment Mode: cash	<div></div>
chocolate - \$10.99 Category: Bag Date: 2024-04-21 Payment Mode: card	<div></div>

Sales Tracker

Item Name

Price

Date

mm/dd/yyyy

Category

Payment Mode

Cash

ADD SALE

Sales on 2024-04-21 | Cash Sales: \$6.99 | Card Sales: \$10.99EXPORT TO PDF

milk - \$6.99 Category: Stationary Date: 2024-04-21 Payment Mode: cash	<div></div>
chocolate - \$10.99 Category: Bag Date: 2024-04-21 Payment Mode: card	<div></div>

Updating a card transaction:

Sales Tracker

Item Name

chocolate

Price

20.99

Date

04/21/2024

Category

Bag

Payment Mode


Card

UPDATE SALE

Sales on 2024-04-21 Cash Sales: \$6.99 Card Sales: \$10.99		EXPORT TO PDF
milk - \$6.99 Category: Stationary Date: 2024-04-21 Payment Mode: cash		
chocolate - \$10.99 Category: Bag Date: 2024-04-21 Payment Mode: card		

Sales Tracker


Item Name



Price

Date

mm/dd/yyyy



Category

Payment Mode

Cash



ADD SALE

Sales on 2024-04-21 | Cash Sales: \$6.99 | Card Sales: \$20.99

EXPORT TO PDF



milk - \$6.99

Category: Stationary | Date: 2024-04-21 | Payment Mode: cash

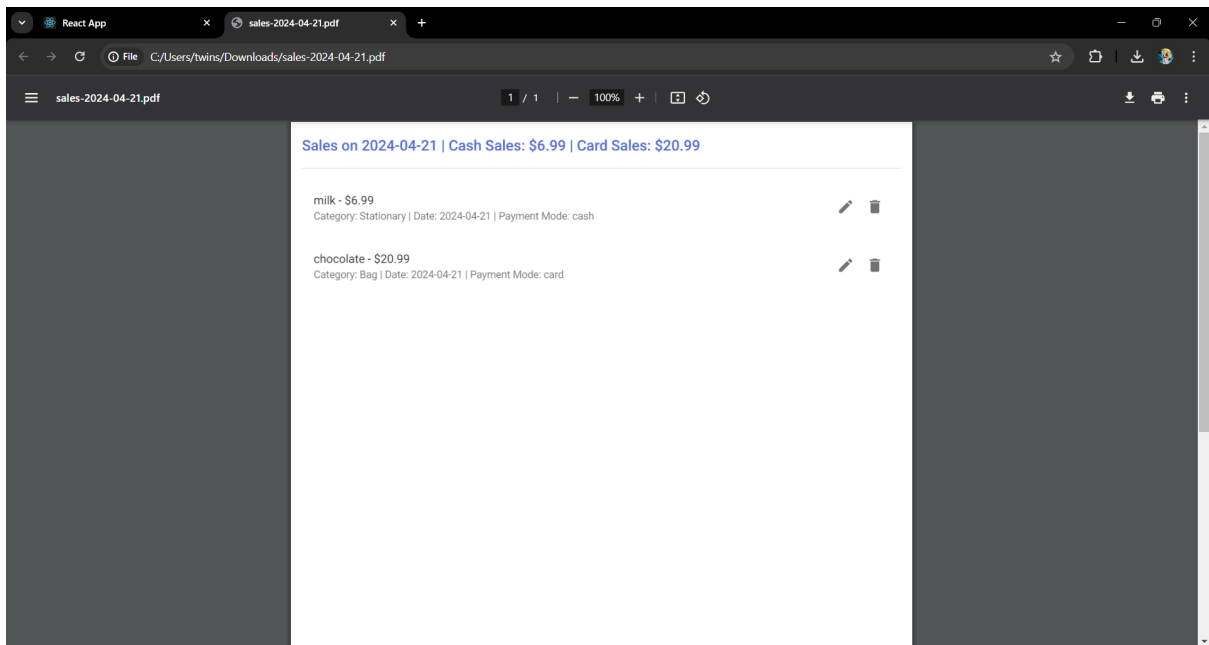


chocolate - \$20.99

Category: Bag | Date: 2024-04-21 | Payment Mode: card



Exporting to PDF:



Deleting a cash transaction:

Sales Tracker

Item Name

Price

Date

mm/dd/yyyy

Category

Payment Mode

Cash

ADD SALE

Sales on 2024-04-21 | Cash Sales: \$6.99 | Card Sales: \$20.99

EXPORT TO PDF

milk - \$6.99

Category: Stationary | Date: 2024-04-21 | Payment Mode: cash

chocolate - \$20.99

Category: Bag | Date: 2024-04-21 | Payment Mode: card

Sales Tracker

Item Name

Price

Date

mm/dd/yyyy

Category

Payment Mode

Cash

ADD SALE

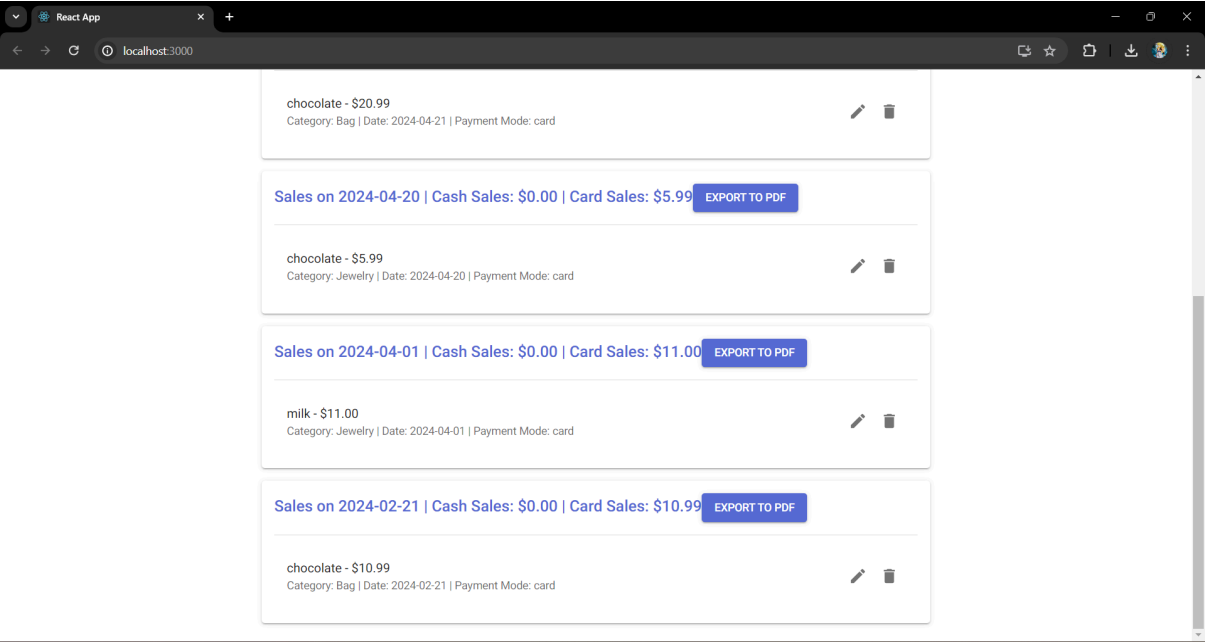
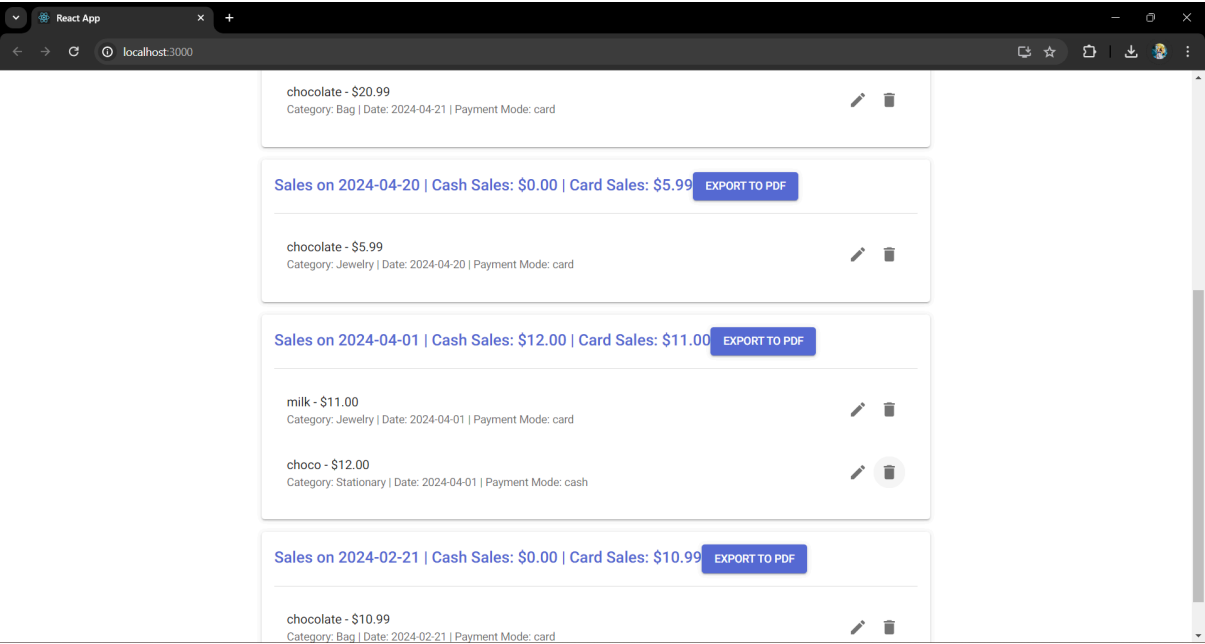
Sales on 2024-04-21 | Cash Sales: \$0.00 | Card Sales: \$20.99

EXPORT TO PDF

chocolate - \$20.99

Category: Bag | Date: 2024-04-21 | Payment Mode: card

Deleting a card transaction:



Sales on different dates:

Sales Tracker

Item Name

Price

Date

mm/dd/yyyy

Category

Payment Mode

Cash

ADD SALE

Sales on 2024-04-21 | Cash Sales: \$0.00 | Card Sales: \$20.99

EXPORT TO PDF

chocolate - \$20.99

Category: Bag | Date: 2024-04-21 | Payment Mode: card

Sales on 2024-04-20 | Cash Sales: \$0.00 | Card Sales: \$5.99

EXPORT TO PDF

chocolate - \$5.99

Category: Jewelry | Date: 2024-04-20 | Payment Mode: card

Sales on 2024-04-01 | Cash Sales: \$0.00 | Card Sales: \$11.00

EXPORT TO PDF

milk - \$11.00

Category: Jewelry | Date: 2024-04-01 | Payment Mode: card

Sales on 2024-02-21 | Cash Sales: \$0.00 | Card Sales: \$10.99

EXPORT TO PDF

Github link -

<https://github.com/SrinivasuluPadigay/itcs4102-5102-sales-tracker>

Conclusion:

Sales Tracker project embodies a groundbreaking approach to sales management, poised to revolutionize business operations through its innovative features and user-centric design. With a keen understanding of the dynamic landscape of modern commerce, our team has crafted a solution that addresses the critical need for efficient sales tracking and optimization. By leveraging cutting-edge technologies, including a React frontend and a GoLang backend, we have developed a platform that not only streamlines sales processes but also empowers users with actionable insights and data-driven decision-making capabilities.

The motivation behind our project stems from a deep-seated recognition of the pivotal role that sales management plays in driving sustainable growth and success in today's competitive market environment. Businesses face myriad challenges in managing sales activities, nurturing customer relationships, and analyzing sales data effectively. Our Sales Tracker project aims to address these challenges head-on by providing a centralized platform that integrates seamlessly with existing workflows, thereby eliminating inefficiencies, breaking down data silos, and unlocking new opportunities for revenue growth.

Throughout the development process, we have adhered to a set of guiding principles aimed at delivering a solution that surpasses the capabilities of existing sales tracking systems. The emphasis on usability and intuitiveness ensures that our platform is accessible to users of all skill levels, reducing the learning curve and increasing adoption rates. Dynamic visualization and automated analysis empower users with real-time insights into sales performance, enabling timely decision-making and proactive management strategies. Moreover, adaptive recommendations based on data analysis help users optimize sales strategies and drive revenue growth effectively.

One of the key strengths of our platform lies in its ability to streamline data input, automate analysis, and provide intelligent notifications, thereby minimizing manual effort and enhancing operational efficiency. By seamlessly integrating with existing systems, our platform ensures compatibility and interoperability, enabling a smooth transition and minimal disruption to established workflows. Additionally, the robust architecture and scalability of our solution position it as a reliable and future-proof investment for businesses of all sizes.

In conclusion, the Sales Tracker project represents a paradigm shift in sales management technology, offering a comprehensive solution that addresses the evolving needs of businesses in the digital age. By harnessing the power of technology and strategic insights, we have developed a platform that empowers organizations to thrive in an increasingly competitive marketplace, driving growth, and success in the journey ahead.