A computer monitor with text on it

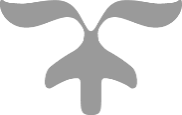
AI-generated content may be incorrect.



Assignment 1

CS251

Dr. Mohammed elramly



Loai Hataba 20230553 Loaiwleed2005@hotmail.com

abdullah mohammed 20230231 abdallamohammmed649@gmail.com

hossam abdelaziz 20230121 hossamabdelaziz2295@gmail.com

**Languages:**

Loai → Java

Abdullah → Java

Hossam → Java

**Learning:**

|  |  |  |
| --- | --- | --- |
| Name | Duration | Sources |
| Loai | 8 Hours | <https://youtube.com/playlist?list=PLJhTWoCm8I6DXaq7XECfyGKtsq4Z6fWZr&si=w_PmOmUhKEH6EyCl>  <https://youtu.be/drQK8ciCAjY?si=cmx5cv4of_BQn4k5> |
| Abdullah | 4 Days | <https://www.tutorialspoint.com/java/index.htm>  <https://www.youtube.com/watch?v=mNvJipMTKSM&list=PLCInYL3l2AajYlZGzU_LVrHdoouf8W6ZN> |
| Hossam | 1 Day | <https://www.youtube.com/watch?v=mNvJipMTKSM&list=PLCInYL3l2AajYlZGzU_LVrHdoouf8W6ZN>https://www.youtube.com/playlist?list=PLJhTWoCm8I6DXaq7XECfyGKtsq4Z6fWZr |

**Food Alternative (App 1 Loai):**

Main Function:

Scanner scanner = new Scanner(System.in);

// Load food from JSON

String jsonPath = "food/foodDictionary.json";

List<FoodItem> foodList = GsonTool.loadFood(jsonPath);

while (true){

            printBanner();

            int menu = optionsMenu(scanner);

            switch(menu)

            {

                // Alternative Food

                case 1:

                    foodMenu(scanner, foodList);

                    int ans = continueApp(scanner);

                    if (ans == 0){

                        scanner.close();

                        System.exit(0);

                    }

                    break;

                //Add new Food

                case 2:

                    addFood(scanner, foodList, jsonPath);

                    int ans2 = continueApp(scanner);

                    if (ans2 == 0){

                        scanner.close();

                        System.exit(0);

                    }

                    break;

                // Delete Food

                case 3:

                    deleteFood(scanner, foodList, jsonPath);

                    int ans3 = continueApp(scanner);

                    if (ans3 == 0){

                        scanner.close();

                        System.exit(0);

                    }

                    break;

                case 4:

                    prinInfoBanner();

                    int ans4 = continueApp(scanner);

                    if (ans4 == 0){

                        scanner.close();

                        System.exit(0);

                    }

                    break;

                case 5:

                    System.out.println("\nGoodbye!!");

                    scanner.close();

                    System.exit(0);

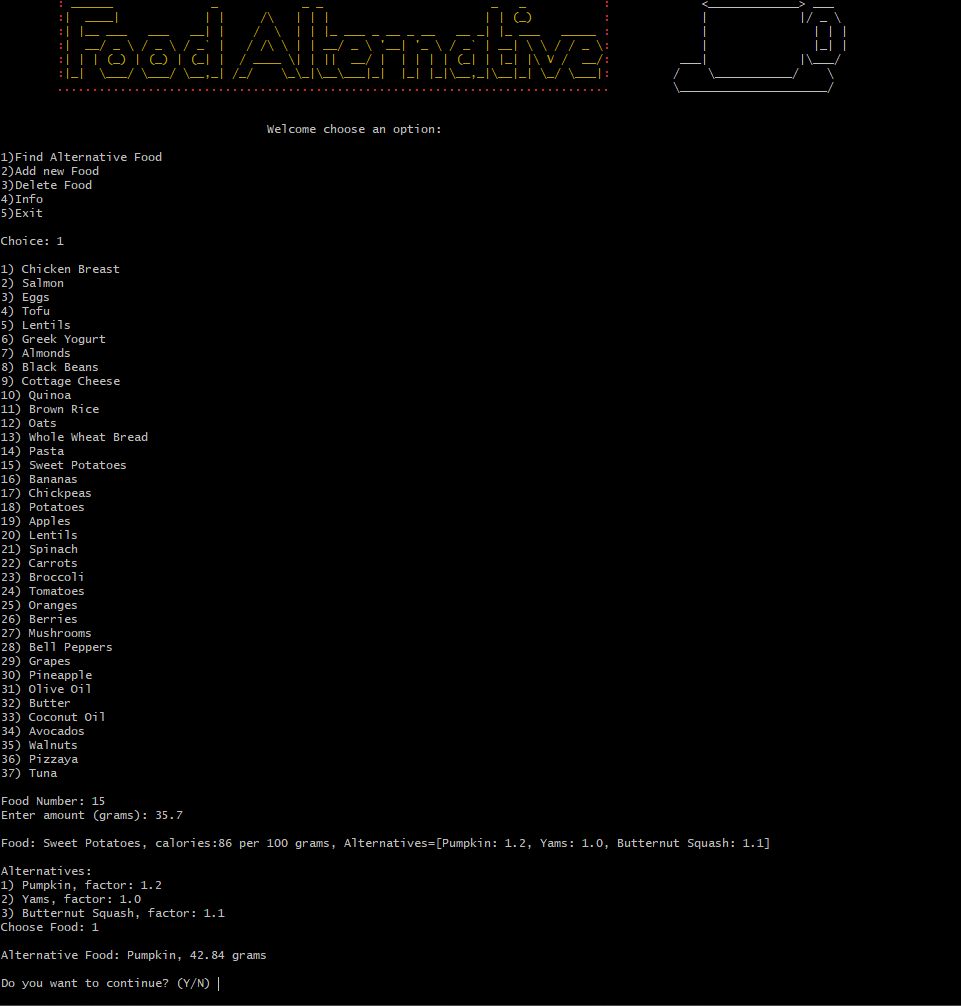
                    break;

            }

        }

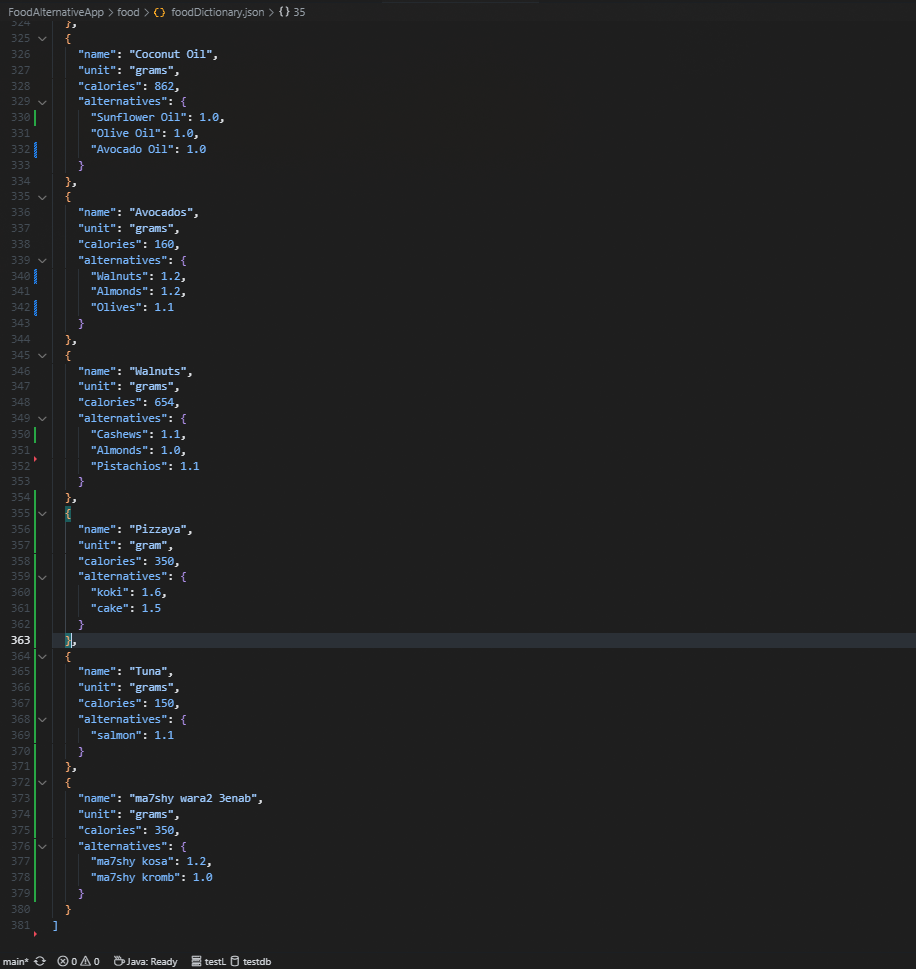
    }

Screenshots:

A screenshot of a computer

AI-generated content may be incorrect.







A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

Video Link:

<https://drive.google.com/file/d/17hZjf0I-YBnwHZ371pdMD7uhwbIYaQP3/view?usp=sharing>

**Budget Tracker (App 2 Abdullah):**

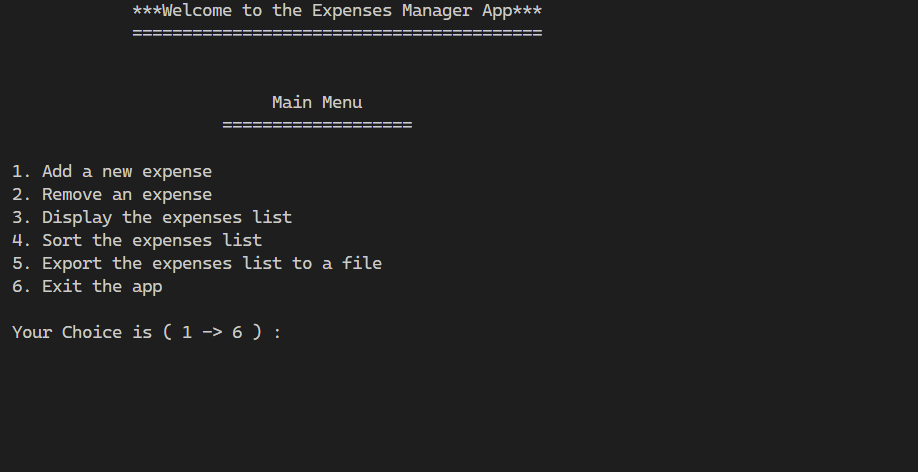
Main Function:

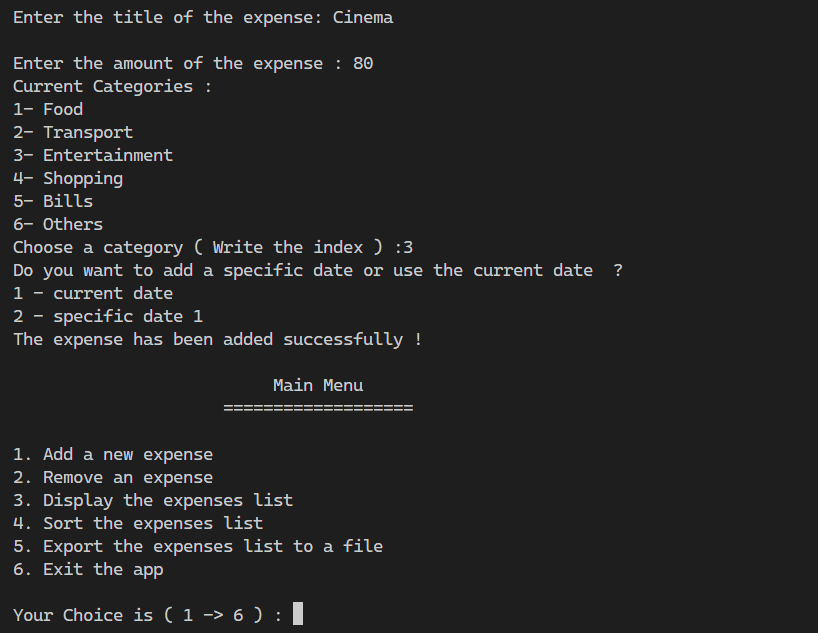
System.out.println(

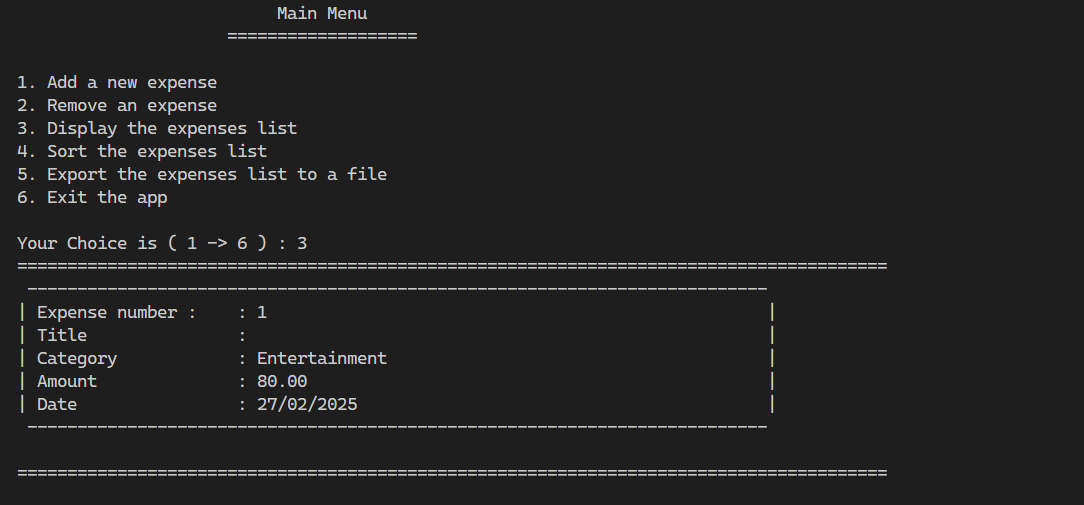
" \*\*\*Welcome to the Expenses Manager App\*\*\*");  
 System.out.println(

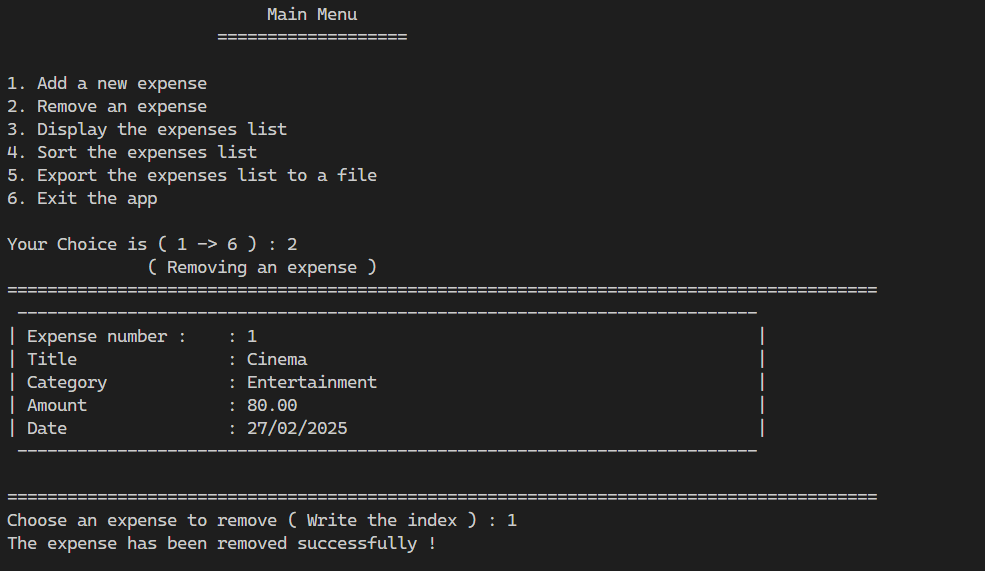
" =========================================\n");  
 // Create an instance of the expenses list   
 final ExpensesList myExpenses = new ExpensesList();  
 //The app menu :   
 while (true) {  
 System.out.println("\n Main Menu ");  
 System.out.println(" ===================\n");  
 // Main menu options :   
 // 1 Adding an expense :   
 System.out.println("1. Add a new expense ");  
 // 2 removing an expense :  
 System.out.println("2. Remove an expense ");  
 // 3 Display the expenses list :  
 System.out.println("3. Display the expenses list ");  
 // 4 Sort the expenses list :   
 System.out.println("4. Sort the expenses list ");  
 // 5 Export the expenses list to a file :  
 System.out.println("5. Export the expenses list to a file ");  
 // 6 Exit the app :  
 System.out.println("6. Exit the app ");  
 //read the user choice :  
  
 final int choice = validInput.getValidInt("\nYour Choice is ( 1 -> 6 ) : ", "Error : Invalid Choice !!", 1, 6);  
 switch (choice) {  
 case 1 ->  
 myExpenses.addExpense();  
 case 2 ->  
 myExpenses.removeExpense();  
 case 3 ->  
 myExpenses.displayExpenses();  
 case 4 ->  
 myExpenses.sortExpenses();  
 case 5 ->  
 myExpenses.exportExpenses();  
 case 6 -> {  
final int ch = validInput.getValidInt("Do you want to saving before closing ?\n1)Yes\n2)No ", "Error : Invalid Choice !!", 1, 2);  
 if (ch == 1) {  
 myExpenses.exportExpenses();  
 }  
 System.out.println("Terminating the program :(");  
 return;  
 }  
 default ->  
 throw new AssertionError();  
 }  
 }

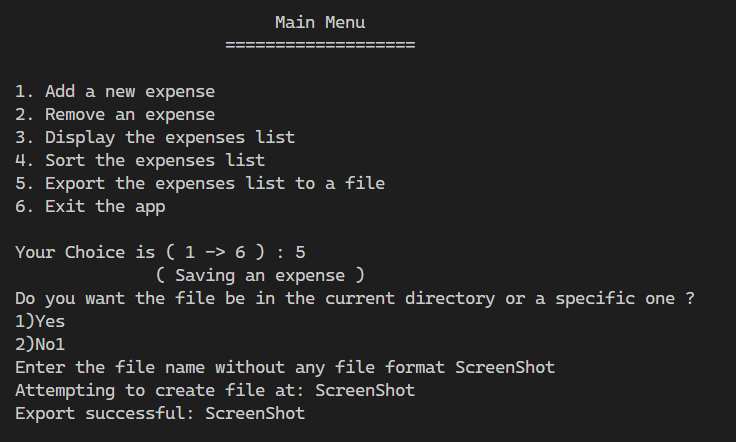
Screenshots:

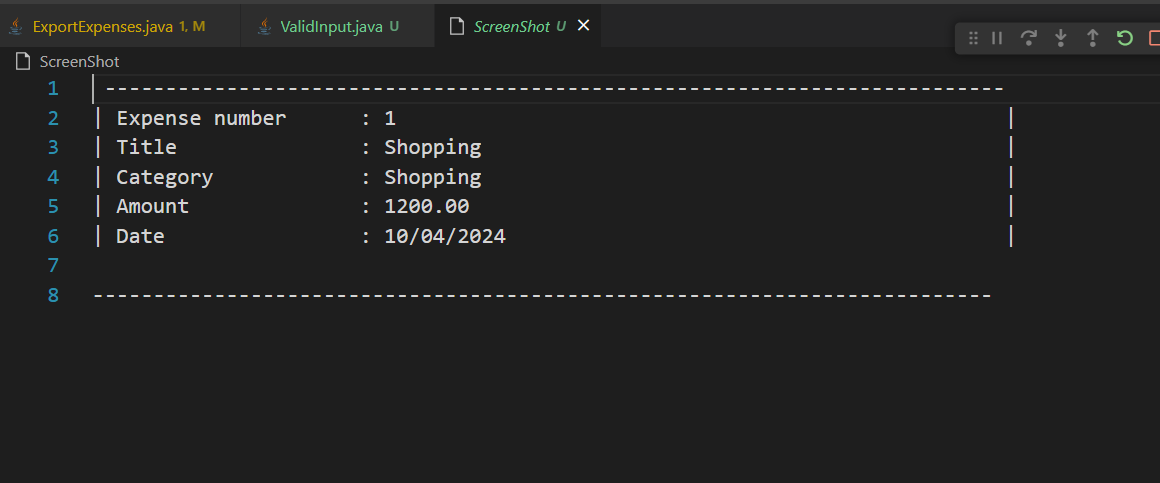












Video Link:

<https://drive.google.com/file/d/1_kMyuxn59d8yfKYCnpGyDzSnsc1SPGdx/view?usp=drive_link>

**Parking System (App 3 Hossam):**

Main Function:

ParkingLot parkingLot = new ParkingLot(10);

Scanner scanner = new Scanner(System.in);

parkingLot.displayGrid();

// Track the last time we checked for expired reservations

long lastReservationCheck = System.currentTimeMillis();

while (true) {

// Check for expired reservations every 5 seconds

long currentTime = System.currentTimeMillis();

if (currentTime - lastReservationCheck > 5000) { // 5 seconds

parkingLot.checkReservations();

lastReservationCheck = currentTime;

}

// Main menu options

System.out.println("" +

"\n1. Park Vehicle" +

"\n2. Remove Vehicle" +

"\n3. Show Parking Status" +

"\n4. Reserve Slot" +

"\n5. View Parking History" +

"\n6. Admin Mode" +

"\n7. Search Vehicle" +

"\n8. Change Parking Rates" +

"\n9. View Statistics" +

"\n10. Exit");

int choice = scanner.nextInt();

scanner.nextLine();

// Check for expired reservations after any user action

parkingLot.checkReservations();

switch (choice) {

case 1:

System.out.print("Enter license plate: ");

String plate = scanner.nextLine();

System.out.print("VIP Slot? (yes/no): ");

boolean isVIP = scanner.nextLine().equalsIgnoreCase("yes");

System.out.print("Vehicle type (car/motorcycle/truck): ");

String vehicleType = scanner.nextLine().toLowerCase();

parkingLot.parkVehicle(plate, isVIP, vehicleType);

parkingLot.displayGrid();

break;

case 2:

System.out.print("Enter license plate to remove: ");

plate = scanner.nextLine();

parkingLot.removeVehicle(plate);

parkingLot.displayGrid();

break;

case 3:

parkingLot.displayGrid();

break;

case 4:

System.out.print("Enter slot number to reserve: ");

int slotNum = scanner.nextInt();

scanner.nextLine();

System.out.print("Enter reservation duration (hours): ");

int hours = scanner.nextInt();

scanner.nextLine();

parkingLot.reserveSlot(slotNum, hours);

break;

case 5:

parkingLot.displayParkingHistory();

break;

case 6:

// Simple password protection for admin mode

System.out.print("Enter admin password: ");

String password = scanner.nextLine();

if (password.equals("hoss123")) {

parkingLot.adminMode();

} else {

System.out.println("Incorrect password!");

}

break;

case 7:

System.out.print("Enter license plate to search: ");

plate = scanner.nextLine();

parkingLot.searchVehicle(plate);

break;

case 8:

// Password protection for changing rates

System.out.print("Enter admin password: ");

password = scanner.nextLine();

if (password.equals("hoss123")) {

System.out.print("Enter new regular rate: ");

double regularRate = scanner.nextDouble();

System.out.print("Enter new VIP rate: ");

double vipRate = scanner.nextDouble();

scanner.nextLine();

ParkingFeeCalc.updateRates(regularRate, vipRate);

System.out.println("Rates updated successfully!");

} else {

System.out.println("Incorrect password!");

}

break;

case 9:

parkingLot.displayStatistics();

break;

case 10:

System.out.println("Exiting...");

return;

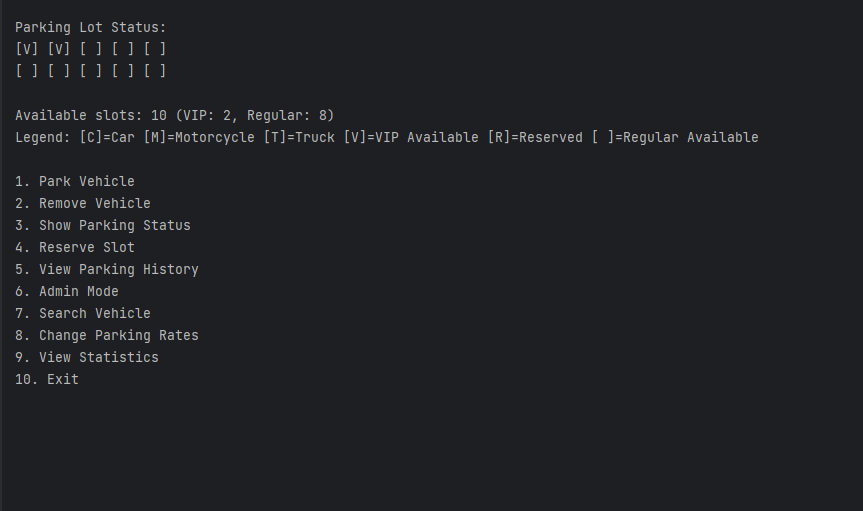
default:

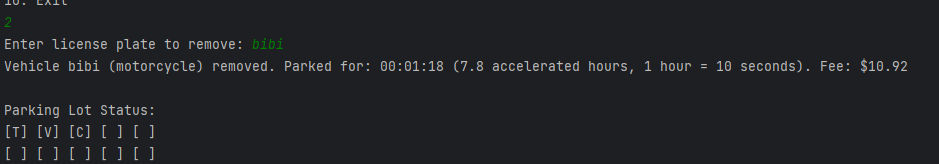
System.out.println("Invalid option!");

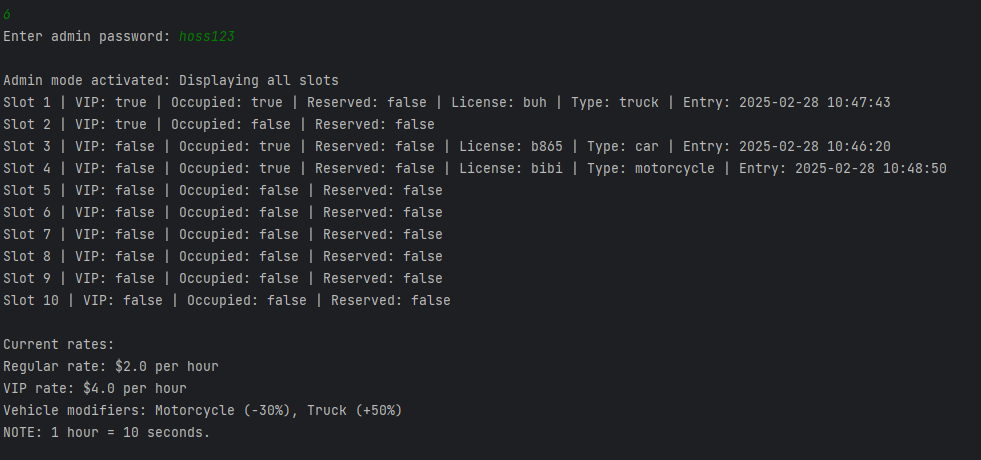
}

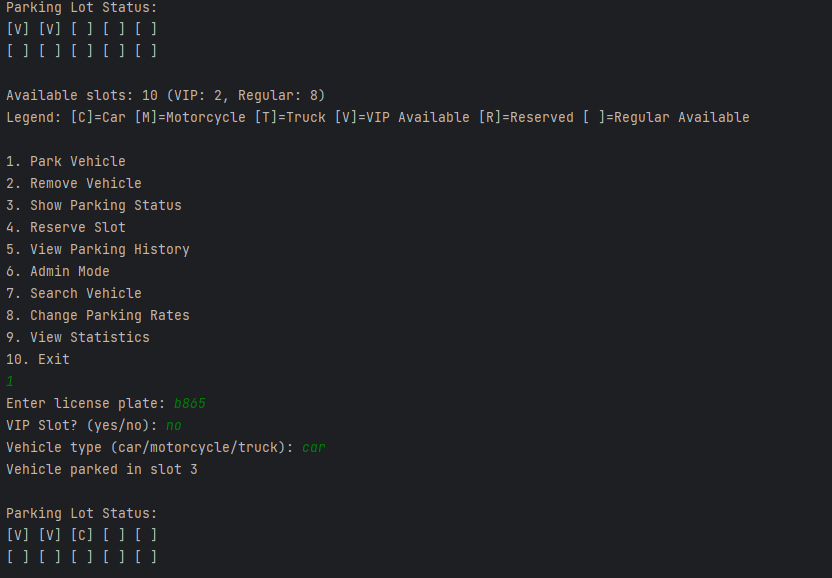
}

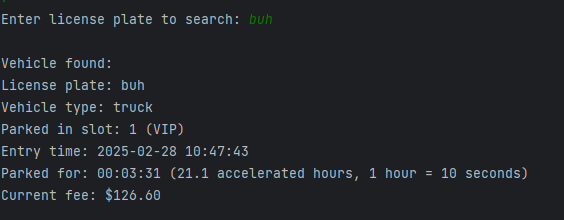
Screenshots:

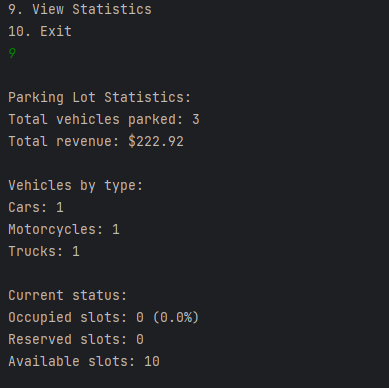












Video Link:

Link:

##### https://drive.google.com/file/d/1E3eVMdre3xwt6Xg1BuPAEF-Zgqf1\_IS\_/view?usp=sharing

**LCNC Analysis: (AppGyver vs Glide)**

**1. Introduction**

Low-code and no-code development platforms have gained popularity for enabling non-developers and businesses to create applications with minimal coding.

Among these platforms, **AppGyver** and **Glide** stand out as powerful tools for building apps efficiently. This report analyzes AppGyver’s and Glide’s usability, benefits, and system quality while comparing them to determine their strengths and suitability for different use cases.

**2. Evaluation of AppGyver**

**Overview**

AppGyver is a professional-grade no-code development platform that allows users to create web and mobile applications without writing traditional code. It is particularly known for its flexibility and extensive customization options.

**Usability and Features**

* **Drag-and-Drop Interface**: Offers an intuitive builder for designing app layouts and functionalities.
* **Extensive Components**: Provides pre-built UI elements and logic modules.
* **Data Integration**: Supports REST APIs, databases, and third-party services.
* **Multi-Platform Deployment**: Applications can be deployed on web, iOS, and Android.
* **Logic and Automation**: Allows users to define workflows and dynamic logic visually.

**Benefits and System Quality**

* **Customization**: Unlike many no-code tools, AppGyver enables deep customization, making it ideal for complex applications.
* **Performance**: Apps built with AppGyver are optimized for high performance, especially on mobile devices.
* **Scalability**: Supports scalable applications, making it suitable for startups and enterprises.
* **Security**: Provides robust authentication and data security features.

**Impact on Developers' Roles**

While no-code platforms like AppGyver simplify application development, they are unlikely to replace traditional developers entirely. Instead, they serve as **enhancement tools** that allow developers to prototype faster and focus on more complex backend logic. Additionally, organizations can leverage no-code platforms for internal tools without needing a dedicated development team.

**3. Evaluation of Glide**

**Overview**

Glide is a no-code development platform that specializes in creating simple, data-driven applications using Google Sheets as a backend. It is widely used for lightweight business applications, internal tools, and prototypes.

**Usability and Features**

* **Google Sheets Integration**: Data is dynamically synced with Google Sheets, making data management straightforward.
* **Pre-Built Templates**: Provides templates to accelerate app creation.
* **Mobile-First Design**: Optimized for mobile applications with responsive design.
* **Drag-and-Drop Builder**: Users can easily add and arrange elements without coding.
* **Limited Logic & Automation**: Basic workflows can be set up, but advanced logic is limited.

**Benefits and System Quality**

* **Ease of Use**: Designed for non-technical users, making app creation accessible to a wide audience.
* **Speed of Development**: Apps can be created and deployed in minutes with minimal effort.
* **Cloud-Based**: Eliminates the need for complex hosting or deployment.
* **Data Management**: Real-time updates with Google Sheets ensure seamless synchronization.

**Impact on Developers' Roles**

Glide significantly lowers the barrier for app creation, allowing businesses to create internal tools without needing a development team. However, its **limited customization and scalability** mean that traditional developers are still essential for building feature-rich, enterprise-level applications. Glide is best suited for rapid prototyping and simple, data-driven applications rather than complex business solutions.

**4. Comparison of AppGyver and Glide**

|  |  |  |
| --- | --- | --- |
| Feature | AppGyver | Glide |
| Ease of Use | Moderate learning curve | Very beginner-friendly |
| Customization | High (supports complex logic and UI design) | Limited (focuses on simplicity) |
| Scalability | Suitable for enterprise applications | Best for small projects and internal tools |
| Pricing | Free for solo developers; enterprise pricing available | Free tier with premium plans for business features |
| Integrations | REST API, third-party services, external databases | Google Sheets, Airtable, Zapier, and limited API support |

**5. Sample To-Do List App**

As part of this comparison, we created a **To-Do List App** using Glide. The application enables users to:

* Add new tasks with descriptions.
* Mark tasks as completed.
* Store and retrieve data using Google Sheets.
* Access the app from both web and mobile devices.

Glide’s simplicity allowed for quick development, making it an excellent choice for basic applications.

Video link:

<https://drive.google.com/file/d/1vqT3N6ripz5jJ3y9BYiqkThsBJVXNKQ7/view?usp=drive_link>

App link:

<https://habit-tracker-app-ills.glide.page>

**6. Conclusion**

AppGyver and Glide both offer unique advantages in the no-code development space. Glide is ideal for quick prototyping and simple applications, while AppGyver provides greater flexibility and customization for more complexprojects.

**Pre-Project :**

**1. Market and Gap analysis**

**Global Practices**

Personal budgeting software isn’t a new ideaa it can be found in many countries, offering comprehensive features to help people manage their finances. An example of those applications include **Mint**, **PocketGuard**, **EveryDollar**, and **GoodBudget**. These apps provide different approaches to budgeting, from detailed manual tracking to automated expense management.

* **Mint**: Automatically syncs with bank accounts and categorizes expenses. It provides visual reports and personalized budgeting recommendations.
* **PocketGuard**: Offers simple budgeting with a focus on preventing overspending by showing how much disposable income is left after accounting for bills and savings.
* **EveryDollar**: Provides zero-based budgeting where users assign every dollar of income to a category, promoting careful financial planning.
* **GoodBudget**: Uses an envelope system where users allocate money to different categories, making it ideal for people who prefer manual control.

**Egypt's Market**

In the Arab region, personal budgeting apps are less prevalent, but there are some local and regional options that cater to Arabic-speaking users. Popular apps include:

* **Masareef**: A budgeting app in Arabic that helps users track income and expenses.
* **Wally**: A widely used app that supports expense tracking and allows users to set financial goals. It supports both English and Arabic languages.
* **Tajer**: Focuses on expense tracking for small businesses but can be used for personal budgeting.
* **Monefy**: An easy-to-use app that allows quick expense logging and provides visual insights.

These apps offer basic features such as expense tracking and simple reporting, but advanced features like bank account integration, financial goal setting, and predictive analytics aren’t really in there.

**Gap Analysis**

Local apps tend to focus only on expense tracking without offering deeper financial insights or integration with banking systems, the apps are fine and do their job but they could really go an extra mile to offer more features in their apps that would be very beneficial for the users such as integrating your bank account so you can track your spendings more accurately.

The key gaps in the Arab region market include:

* **Bank Account Integration**: Very few apps automatically sync with bank accounts or e-wallets, making users manually enter transactions.
* **Financial Goal Setting**: Limited functionality for setting and tracking financial goals like savings for vacations or debt repayment.
* **Customizable Budgets**: Lack of flexible budgeting tools that allow users to set limits for different categories and adjust them dynamically.
* **Insights and Analytics**: Minimal reporting features to help users understand spending patterns or predict future expenses.
* **User Education**: Few apps offer financial education content to help users improve their financial literacy.

**2. Market Segmentation and Research**

**3. Domain Analysis :**

**Introduction:**

This document describes background information that has been gathered about budgeting and how it is handled. This information is to be used to guide the development of software to automate the process of making people track their expenses.

**Glossary:**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Income | Money received from salaries, businesses, investments, or other sources. |
| Expense | Money spent on necessities (rent, food, utilities) and discretionary spending (entertainment, travel) |
| Budget | A financial plan that allocates funds to different spending categories |
| Financial Goal | A target for saving money (e.g., buying a car, going on vacation, repaying loans) |
| Expense Category | A classification for expenses, such as groceries, transportation, education, or entertainment |
| Savings | Money set aside for future use, such as emergency funds or investments |
| Investment | Assets purchased with the goal of generating future income, like stocks or property |
| Bank Integration | The ability to sync financial transactions automatically with bank accounts or e-wallets |
| Zero-Based Budgeting | A method where every dollar is assigned to a specific expense or savings category, leaving no unallocated funds |
| Cash Flow | The movement of money in and out of a person’s finances, tracking income vs. expenses |
| Net Worth | The total value of a person's assets minus liabilities (debts) |
|  |  |
|  |  |

**General Knowledge:**

* Budgets are of different types (see glossary). A budget includes income sources and expenses and may have saving goals for better financial planning.
* Budgets have an owner, usually an individual user, who tracks spending and savings. Each budget also has a title, description, and financial details.
* It is possible to add categories to expenses so that spending groups can be created. This way, users can organize their finances more effectively.
* Outdated budgets or transactions that are no longer relevant can be archived or removed.
* Budgets and financial insights may be viewed by only the user, but there may be options for sharing certain reports with financial advisors or family members.

**Customers and Users:**

* **Potential Clients**: Organizations or businesses that may adopt the software for their customers, such as: Banks offering budgeting tools as a value-added service. Employers providing financial wellness programs for employees. Financial advisory firms help clients with personal finance management.
* **Potential Users:** Individual users are interested in tracking their expenses and managing their monthly budget. These users may include Employees looking to save money from their salary by setting financial goals. Students or young adults managing limited income and expenses. Families planning household budgets and expenses. Freelancers or self-employed individuals tracking irregular income and spending.

**The Environment:**

All users access the budgeting app on their mobile phones, which may run on Android or iOS. Each user has a unique software configuration, with different apps and settings installed on their device. The budgeting app should be compatible with various device configurations and ensure smooth performance across different phone models and operating systems.

**Tasks and procedures currently perform:**

**Input Expenses**:

Users manually enter their daily, weekly, or monthly expenses into the system. Expenses can be categorized (e.g., rent, groceries, entertainment) to track spending patterns more effectively.

**Setting Financial Goals**:

Users define short-term and long-term financial goals, such as saving for a vacation, purchasing a car, or paying off debt. The software helps track progress and provides insights into how much needs to be saved each month to meet the goal.

**Reporting and Insights**:

The app generates financial reports, including spending trends, budget summaries, and savings progress. Visual representations like charts and graphs help users analyze their financial habits and make informed decisions.

**Sync with Bank Account**:

Making the app sync with bank accounts, e-wallets, or credit cards to automatically import transactions. This reduces manual data entry and ensures real-time tracking of expenses and income

**4. Proposed Solution**