Software Design: GotoGro-MRM

Team Details

Team Name:	MSP 14
Tutorial:	Tue 2:30 ATC325
Tutor:	Dr Kaberi Naznin

Members:	
Dylan Jarvis	102093138
Rabya Tayal	103144215
Simon Tran	103602807
Thomas Babicka	103059885
Cody Cronin-Sporys	103610020
Nicholas Dyt	101624265

Software Design

The UML diagram in **Figure 1** displays how the different screens interact with each other. In our model, the UI components are intrinsically linked with the code. For example, each function is called by a button or input field. This is distinctly object oriented where the UI components could be considered objects containing each linking function. These are abstracted and compartmentalised and all exist within another object being the given page the user is looking at. The link between the pages is the universal navigation bar, much like the nav bar of a website. The separation of these pages supports good design principles as they are self-contained and not reliant on any active information other than the shared database.

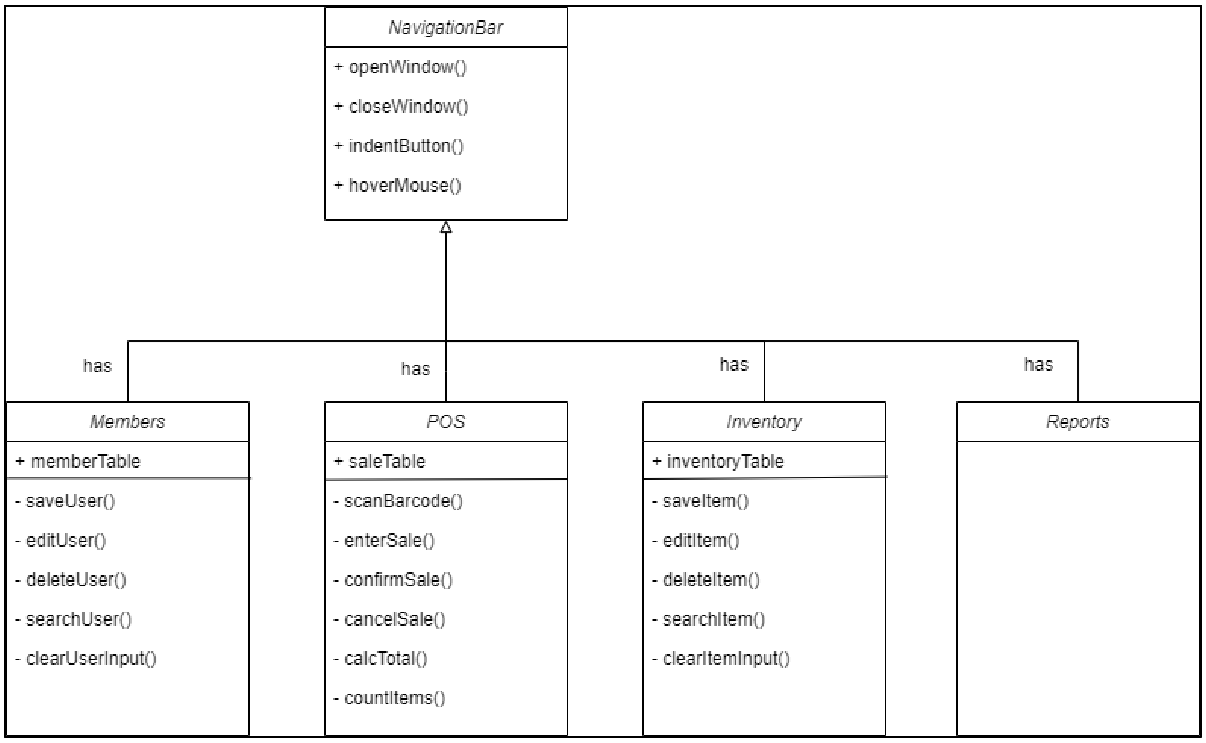

Figure 1: *Basic UML of Goto-Gro Software.*

Software Design Principles

Table 1. *Software Design Principles*

Principle	**Description**
Modularity	The measure of how well the software is broken down into individual components. The components must be separately maintainable from other components and cannot affect others.
High Cohesion & Low Coupling	**High Cohesion** refers to how a single classes function relate to one another. **Low Coupling** refers to classes having the least number of dependencies on other classes.
DRY (Don't Repeat Yourself)	All pieces of code are not repeated in other elements. For example, a function/definition is only defined once. This helps make maintainable and scalable coding.
MVC Design Pattern	Model View Controller (MVC) is a design principle that breaks down each functional component of an application into three separate components. These three components describe the interaction between a user and the backend of an application. A User uses the Controller which in turn manipulates the Model and updates the View that is seen by the User. This is essentially a feedback loop for user input and data manipulation.

Table 2. *Justification of Design Principles*

Principle	**Justification**
Modularity	Each section within the project is separate from another. Changing anything on the 'members page' cannot change or affect the performance of any other page or function. All specific member functions are defined within the member's page. This applies for both the POS and inventory pages. Common SQL statements are defined in the inbuilt 'named queries' section so they can be easily called from anywhere within the project. But they are also only used on their respective pages. The design of this project is fully modular.
High Cohesion & Low Coupling	Each object within this project is designed to perform a single action. For example, on the members page each button only affects the members SQL table and other elements on that page related to the functionality required for members. This is likewise seen across all other pages.
DRY (Don't Repeat Yourself)	Much like cohesion, functions are only defined exactly where needed and sperate from another. Buttons that require an SQL query simply call the pre-defined query from 'named queries' rather than redefine them again. In some instances, repeating lines of code is unavoidable due to the nature of some of the

	functions but it is still minimal for the entire project. For instance, the same logic is used across both the members and inventory pages but with different variables. There is no way to reduce this.
MVC Design Pattern	This is the exact purpose of this solution, and it follows this exactly. A user uses the inbuilt ignition ‘vision’ module that manipulates the MySQL database which in turn updates the ‘vision’ view the users sees. The ‘vision’ module is both the Controller and the View that the user sees whereas the Model is the MySQL database.