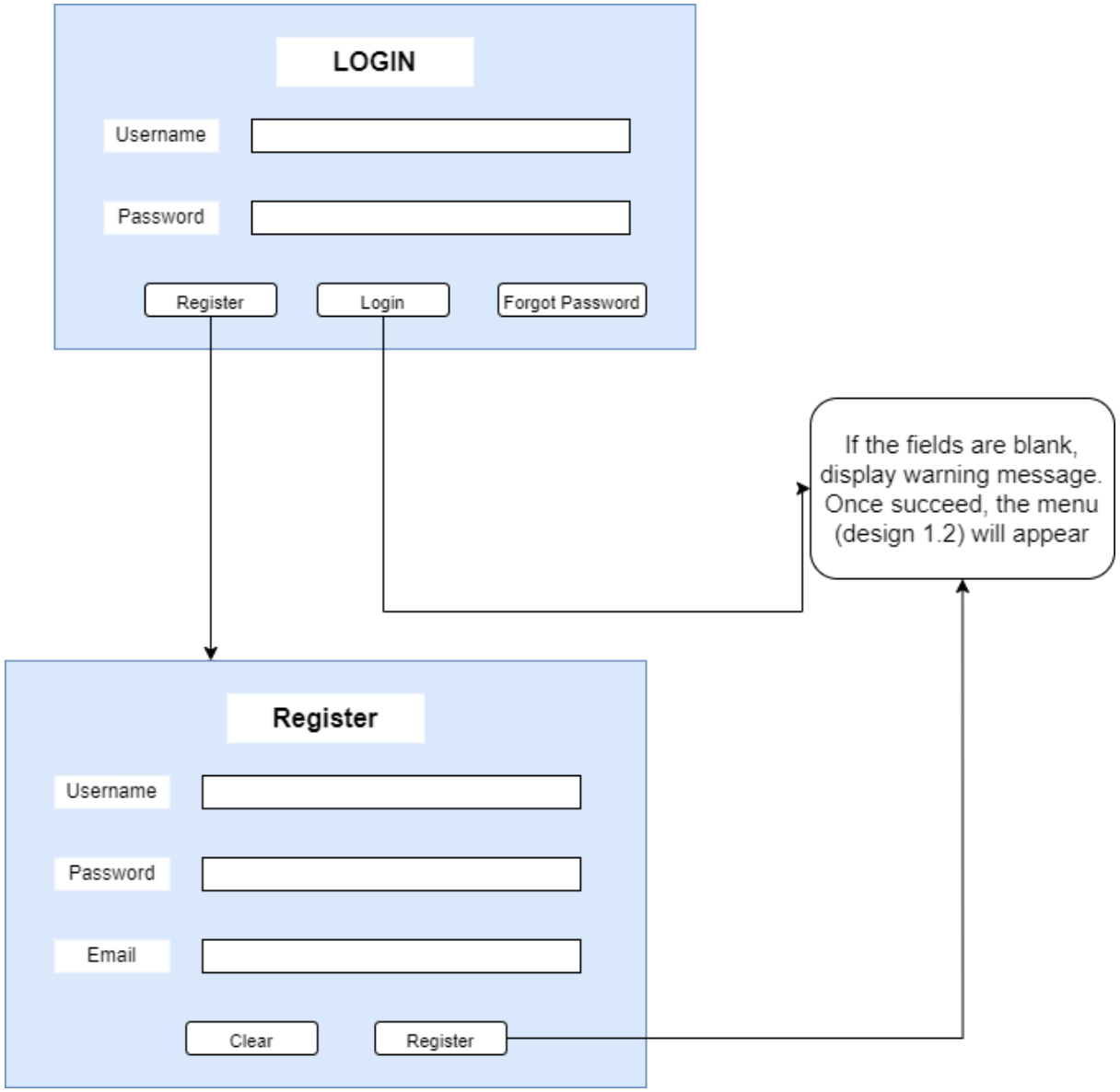
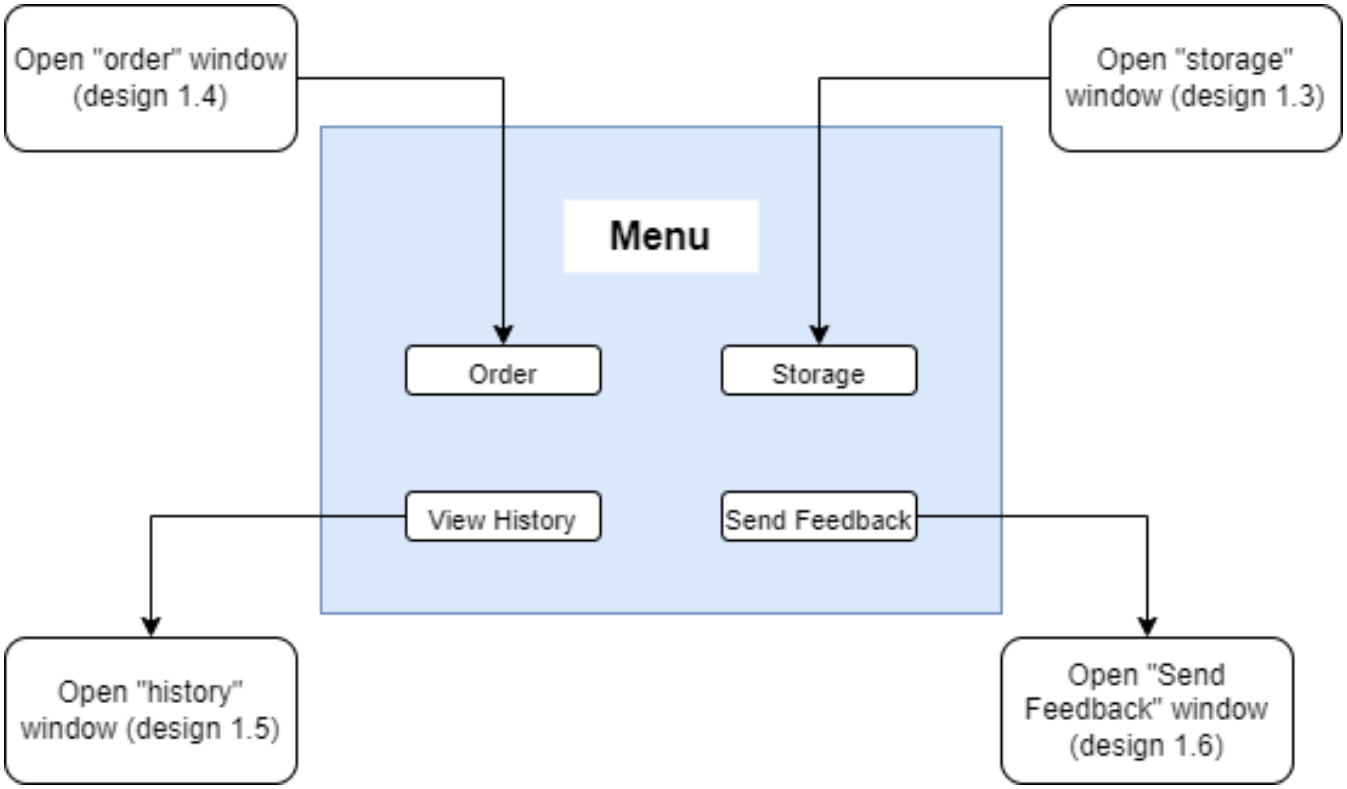
Criteria B: Design Overview (refer to Appendix II, part 1)

DESIGN

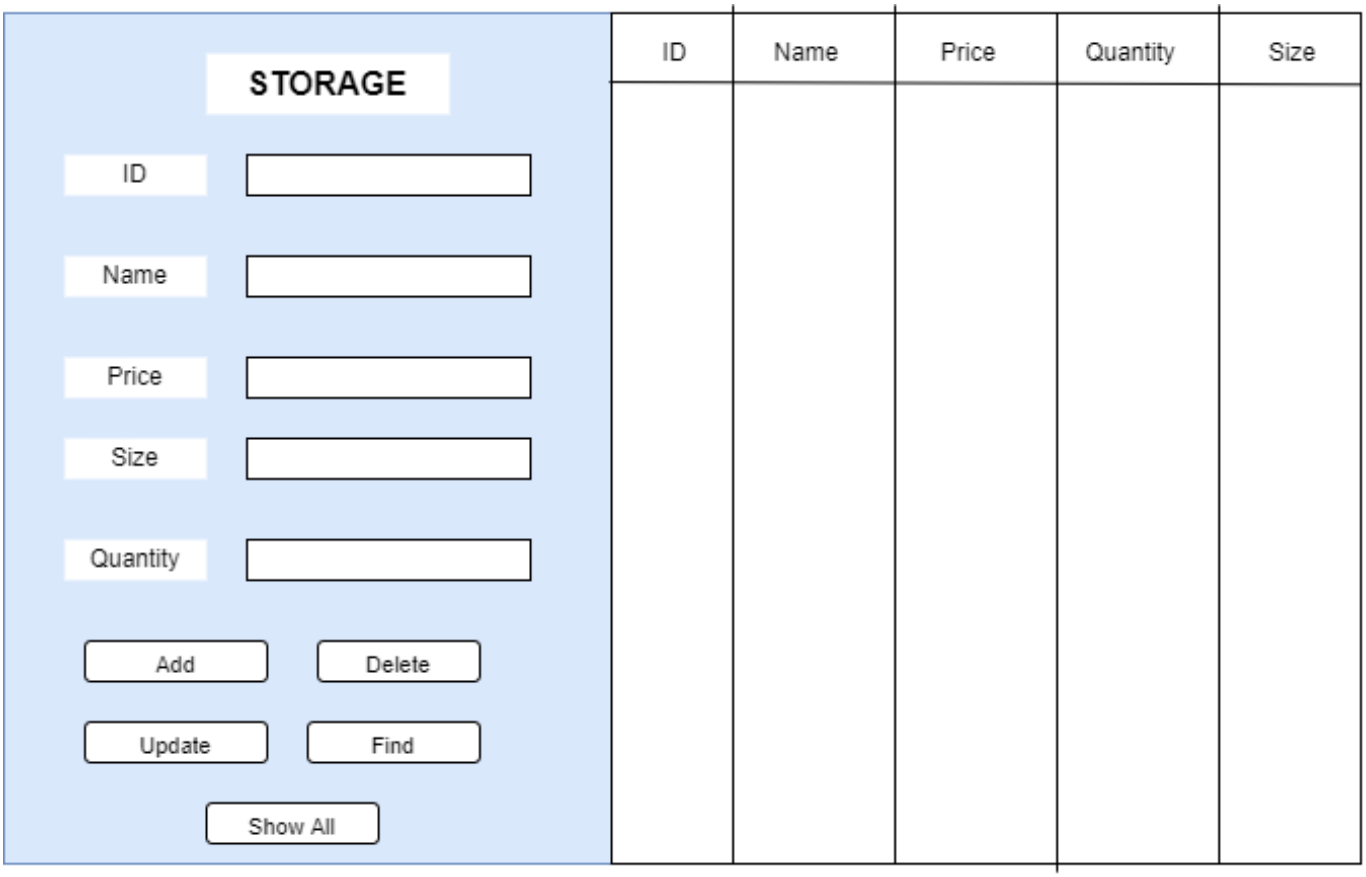
1.1 Login



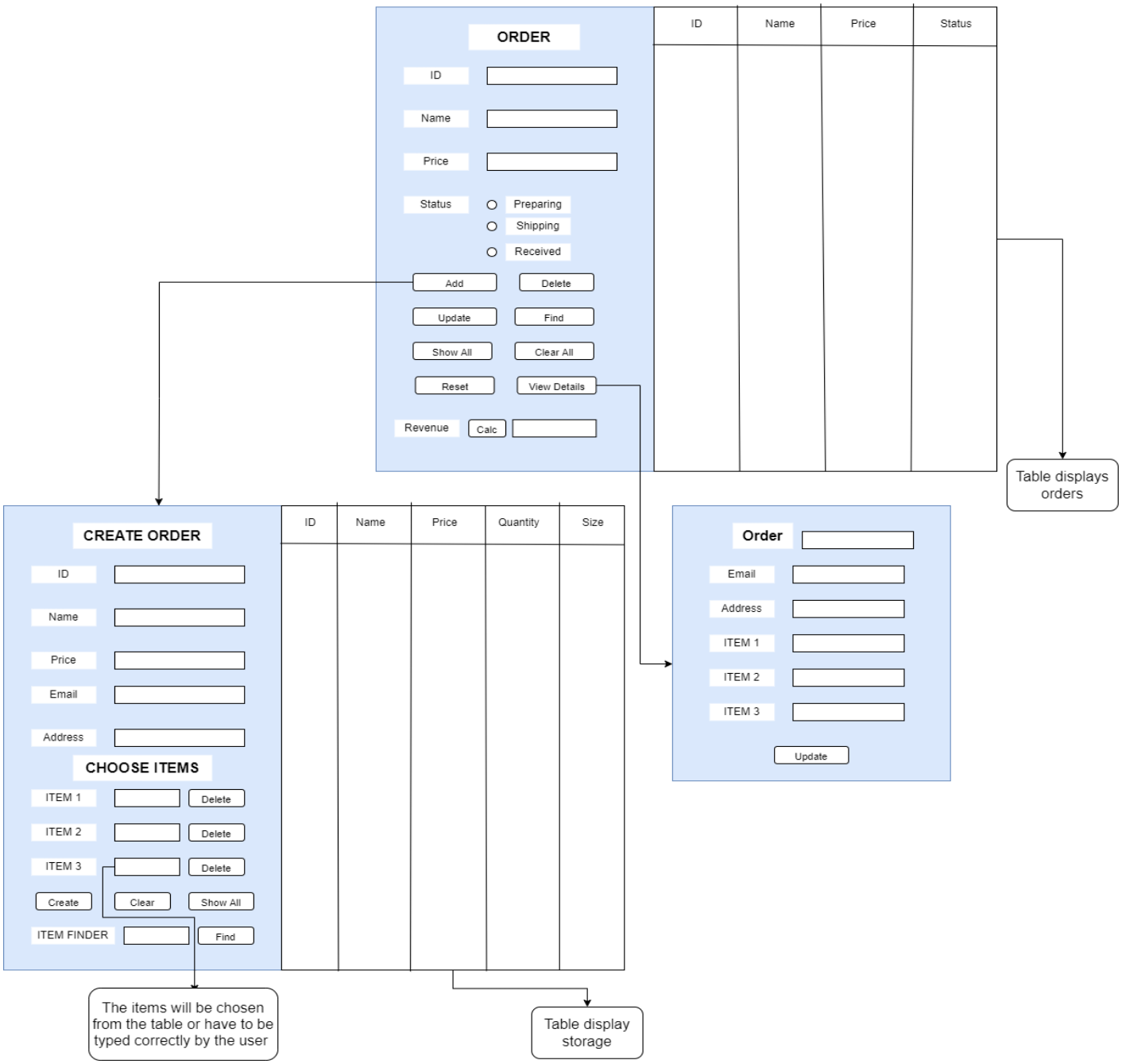
1.2 Menu


Open "order" window (design 1.4)
Open "storage" window (design 1.3)
Menu
Order
Storage
View History
Send Feedback
Open "history" window (design 1.5)
Open "Send Feedback" window (design 1.6)

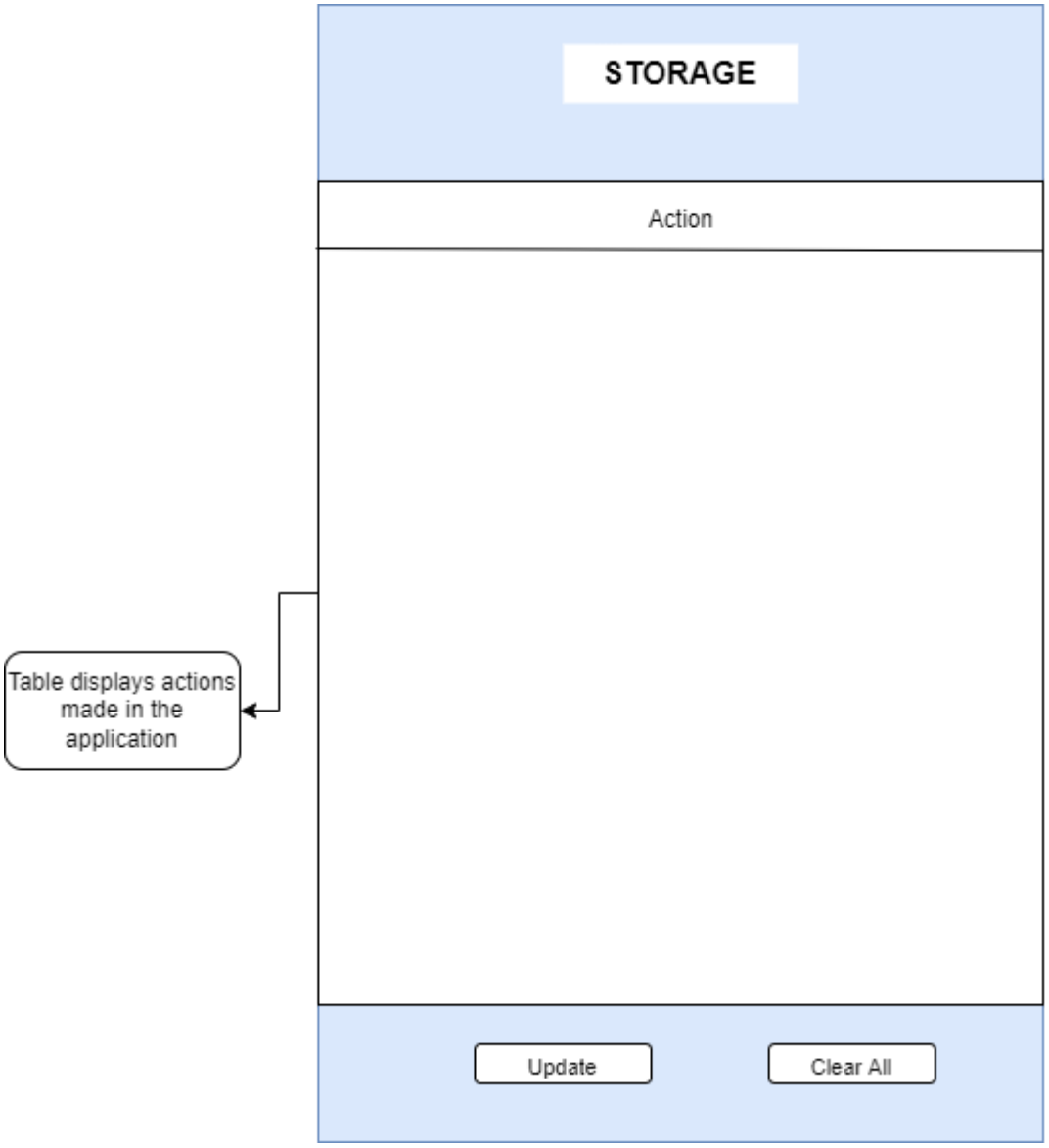
1.3 Storage


STORAGE
ID
Name
Price
Size
Quantity
Add
Delete
Update
Find
Show All
ID
Name
Price
Quantity
Size

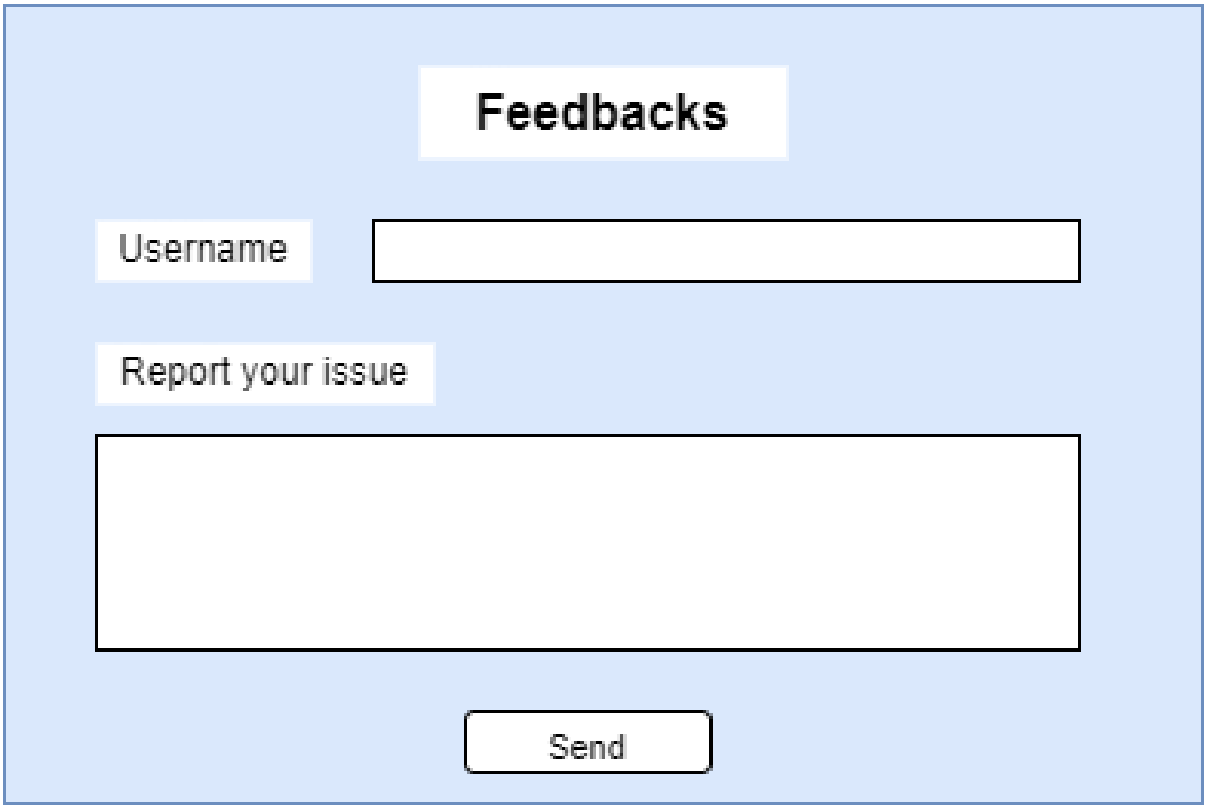
1.4 Order


ORDER
ID
Name
Price
Status
Preparing
Shipping
Received
Add
Delete
Update
Find
Show All
Clear All
Reset
View Details
Revenue
Calc
ID
Name
Price
Status
Table displays orders
CREATE ORDER
ID
Name
Price
Email
Address
CHOOSE ITEMS
ITEM 1
Delete
ITEM 2
Delete
ITEM 3
Delete
Create
Clear
Show All
ITEM FINDER
Find
ID
Name
Price
Quantity
Size
Order
Email
Address
ITEM 1
ITEM 2
ITEM 3
Update
The items will be chosen from the table or have to be typed correctly by the user
Table display storage

1.5 History

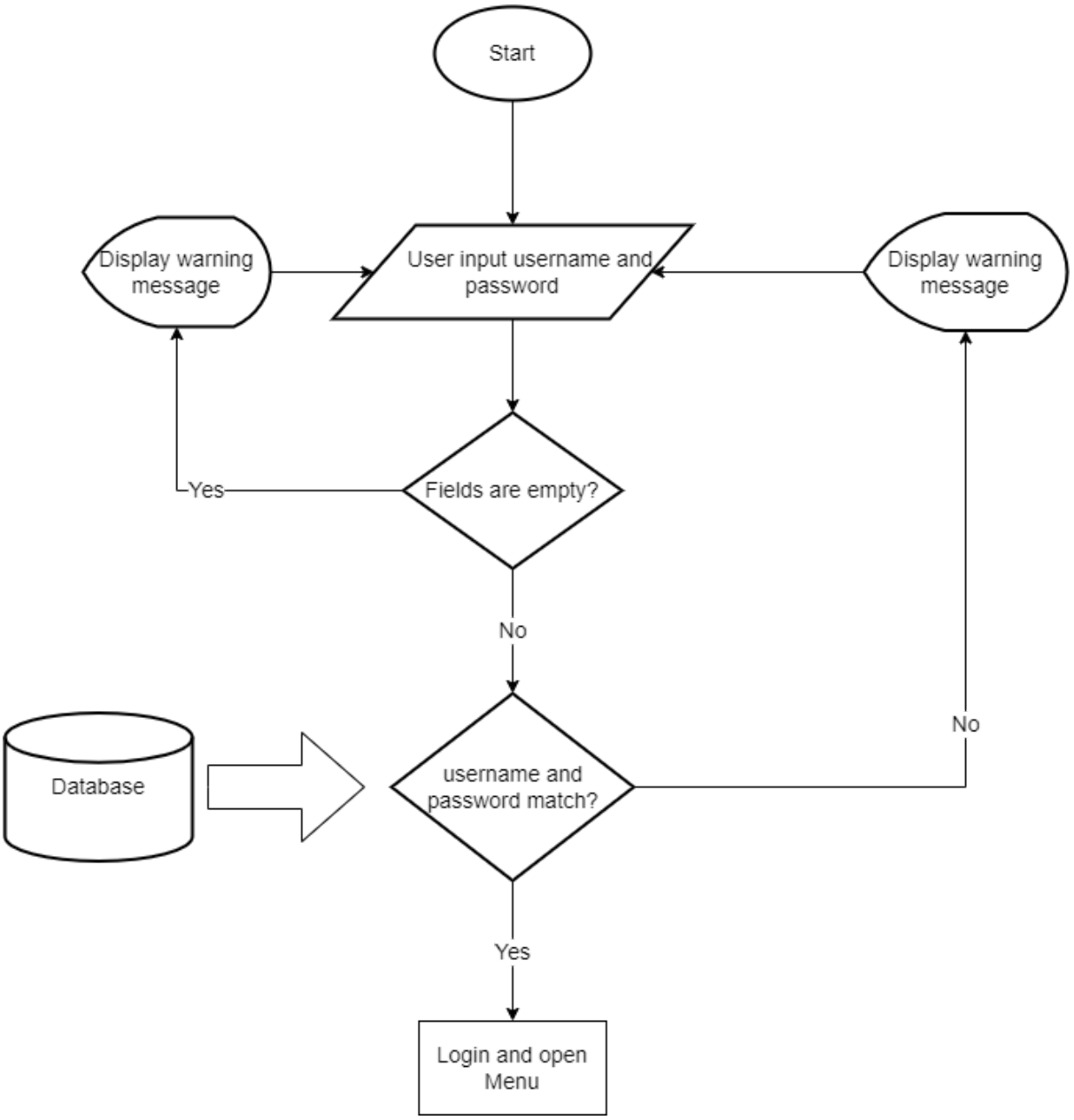

STORAGE
Action
Table displays actions made in the application
Update
Clear All

1.6 Send feedback

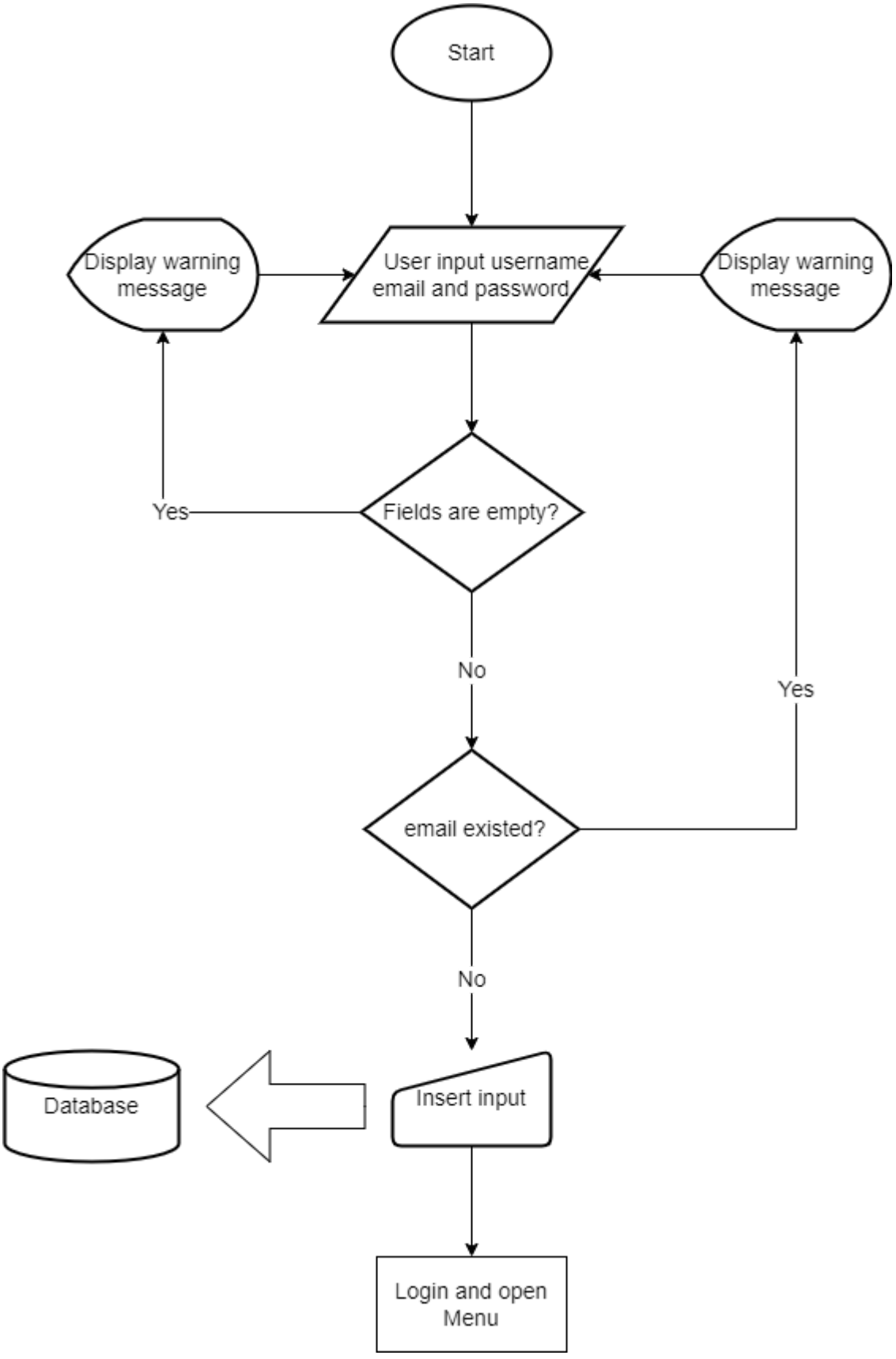


Flowcharts

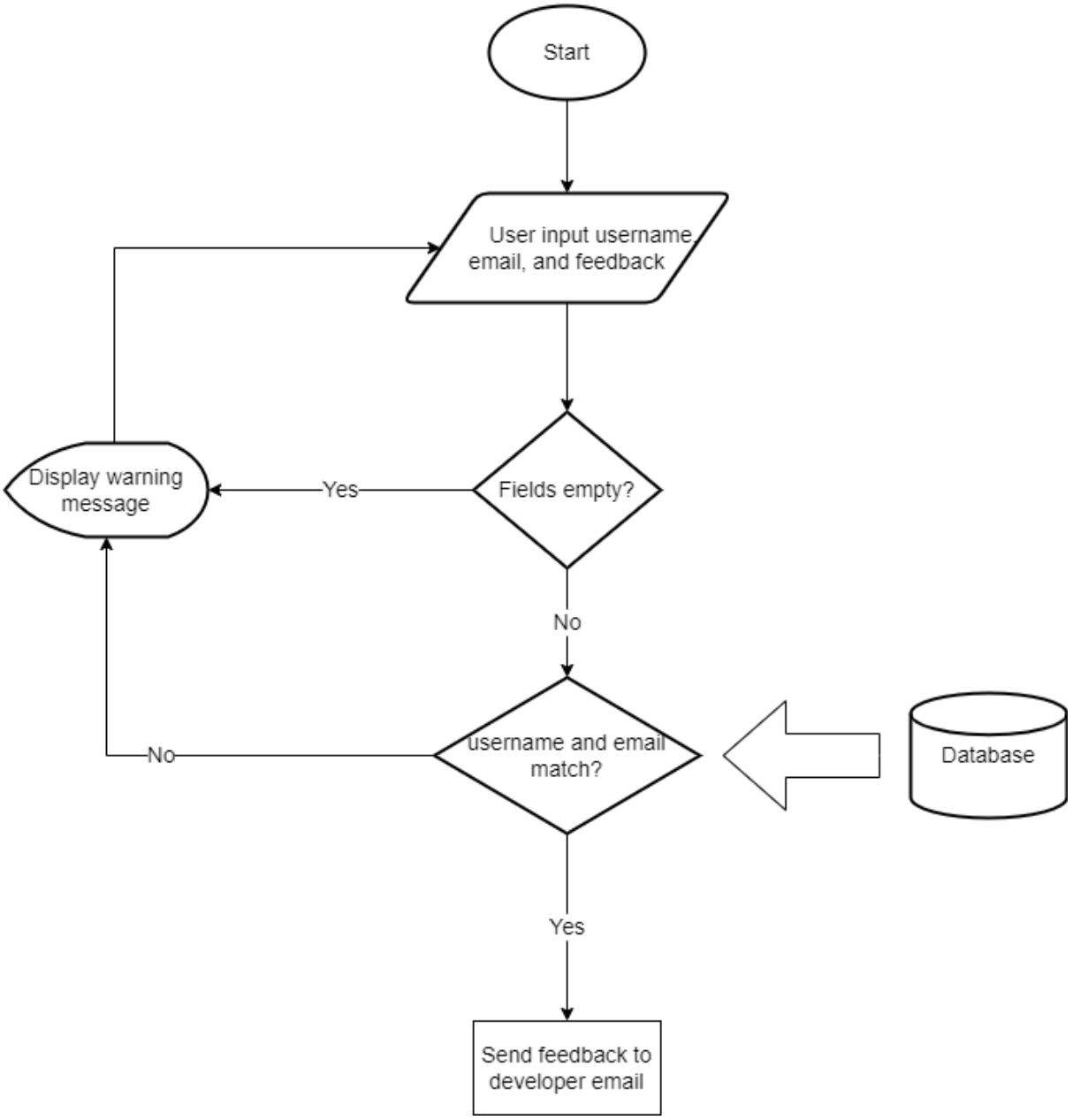
1.1 Login Flowchart



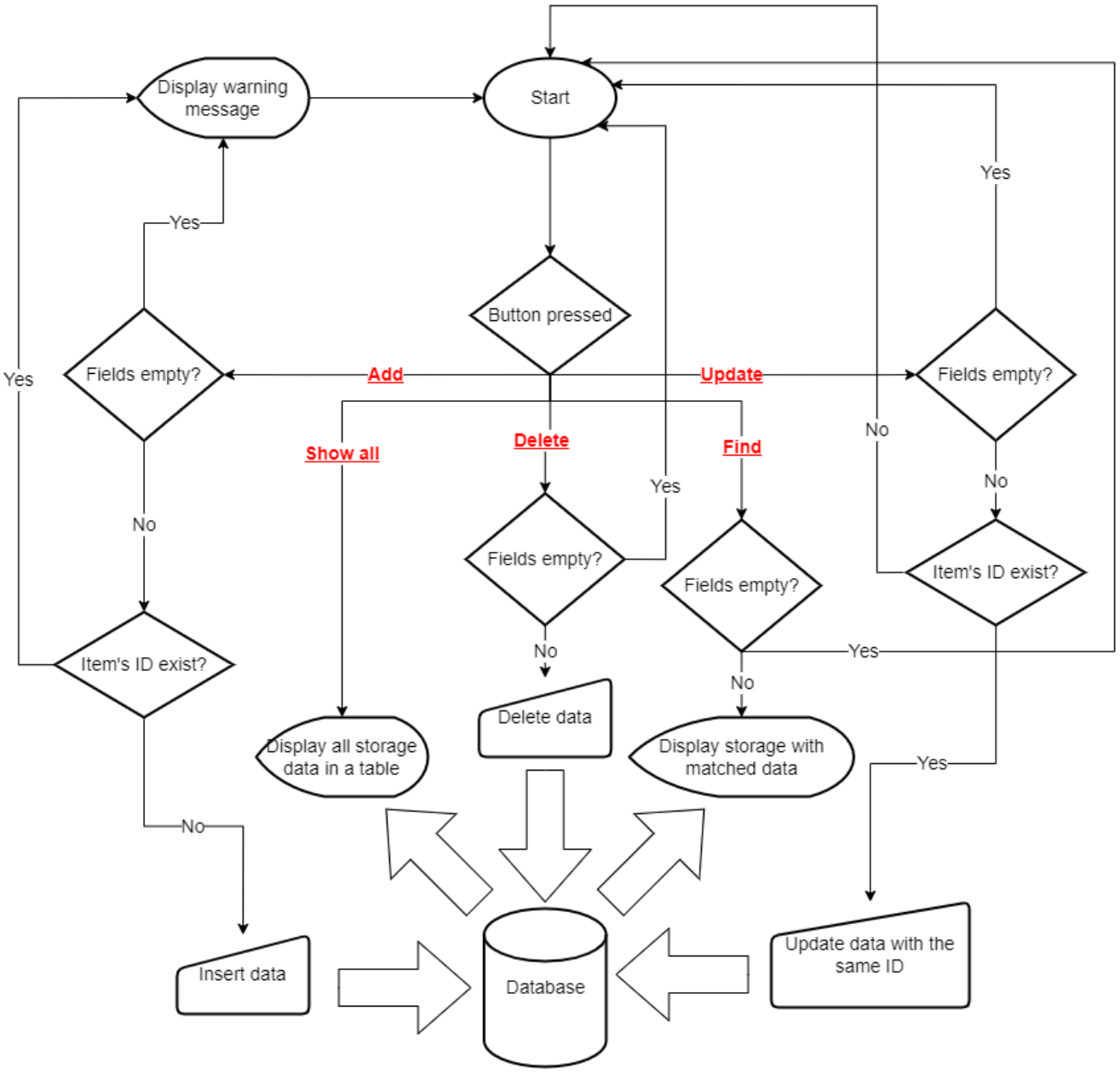
1.2 Register Flowchart



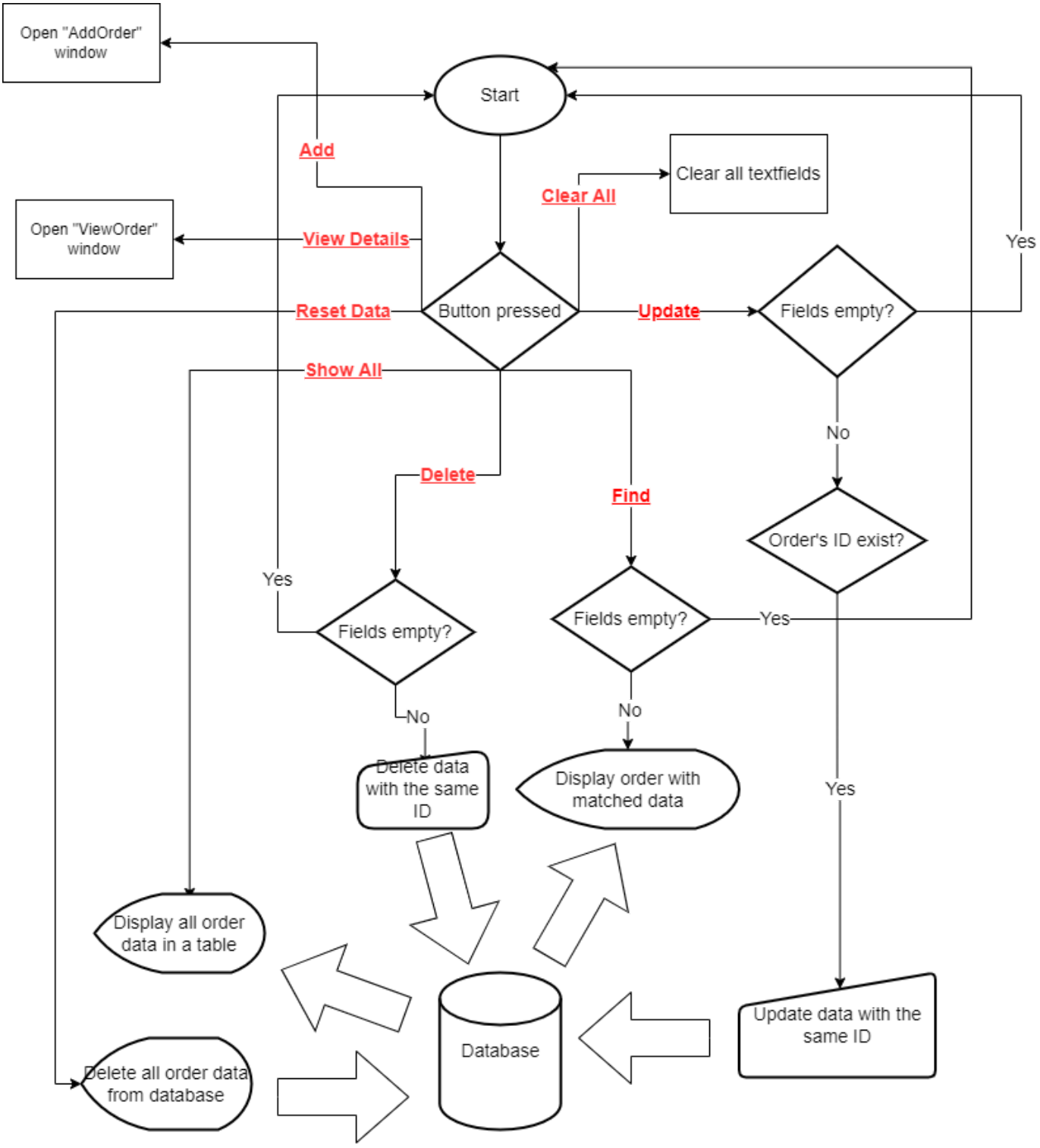
1.3 Send Feedbacks Flowchart



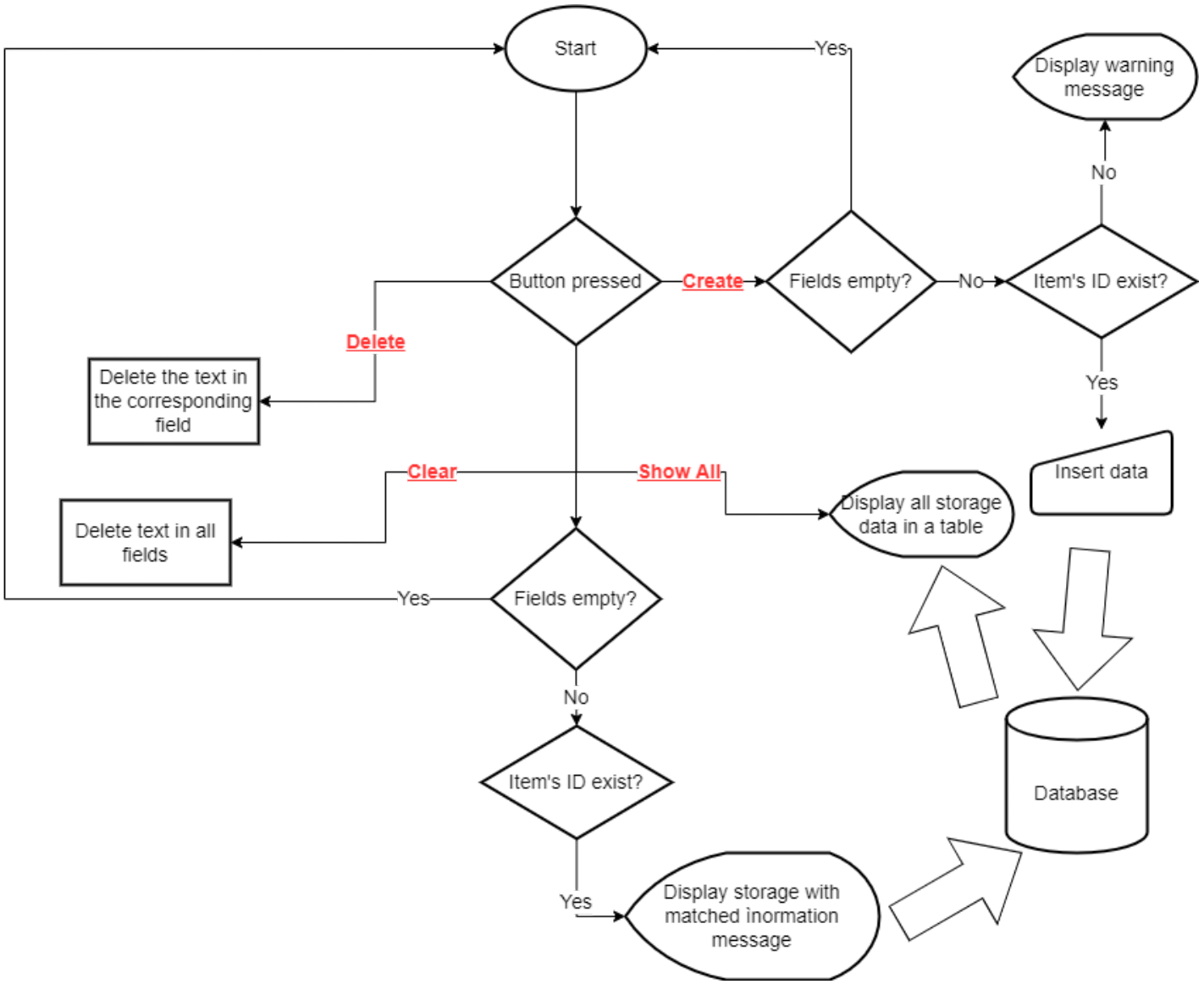
1.4 Storage flowchart



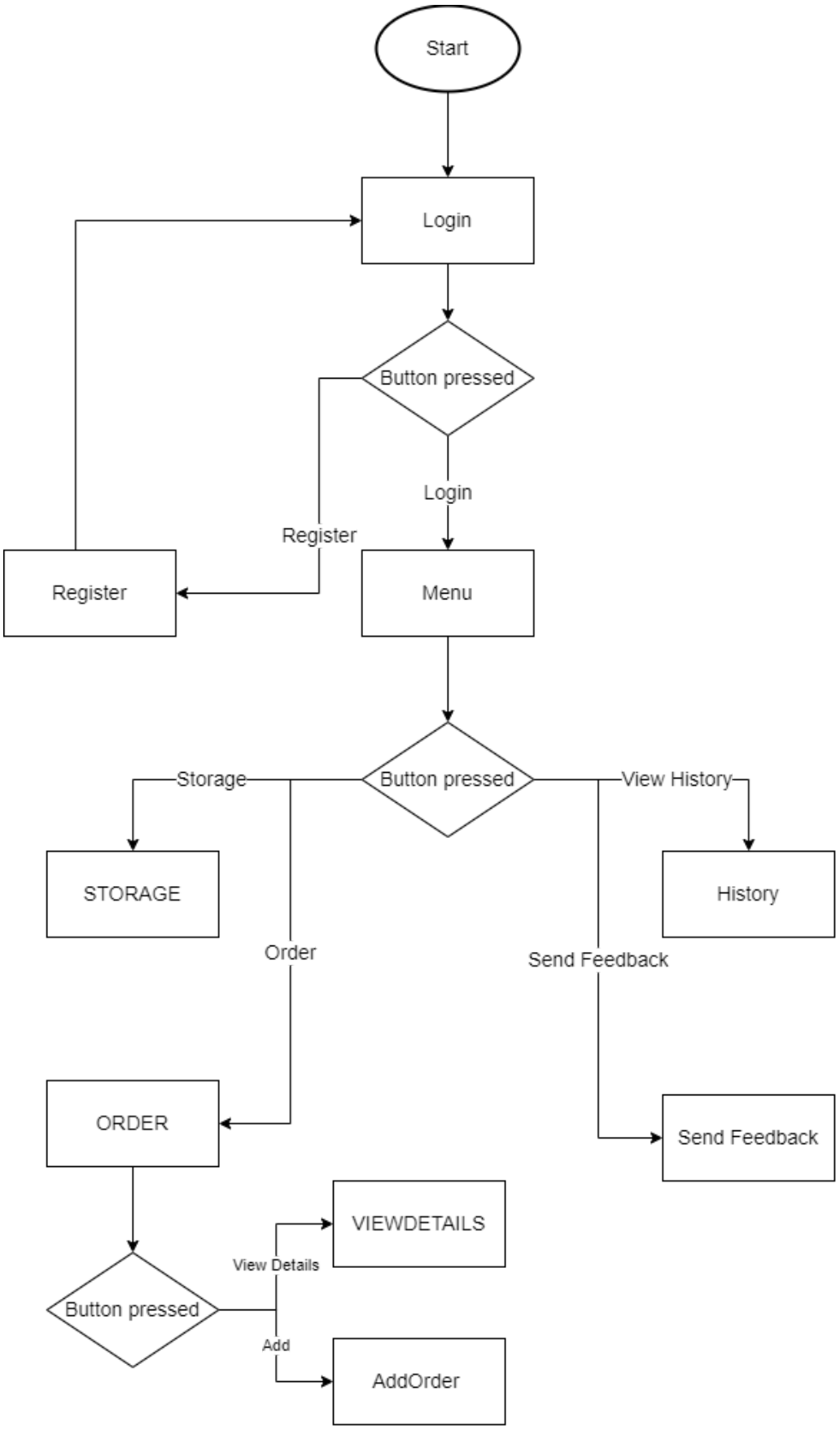
1.5 Order flowchart



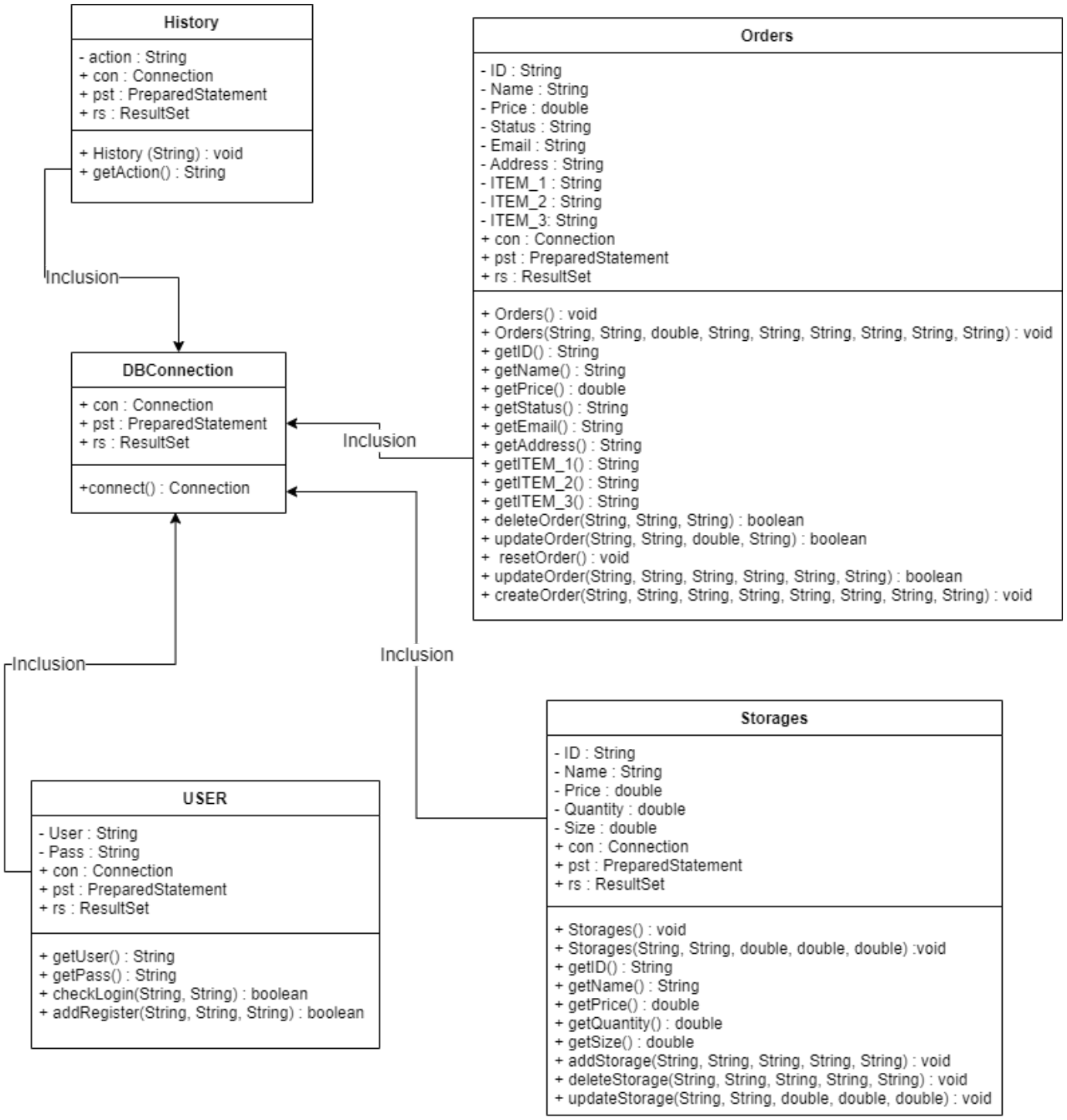
1.6 Add Order Flowchart



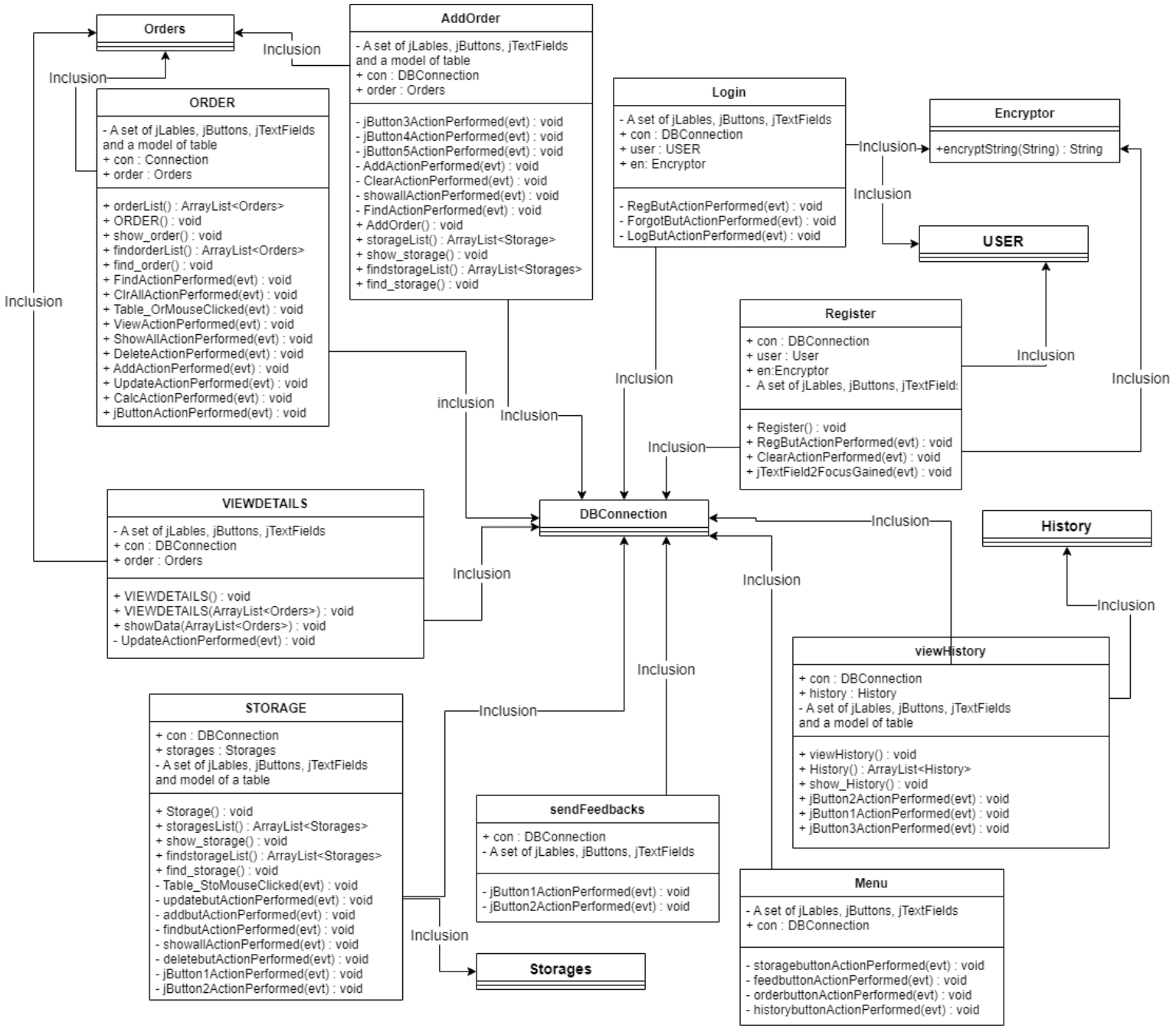
1.7 Interface Flowchart



UML Classes


History
- action : String
+ con : Connection
+ pst : PreparedStatement
+ rs : ResultSet
+ History (String) : void
+ getAction() : String
Orders
- ID : String
- Name : String
- Price : double
- Status : String
- Email : String
- Address : String
- ITEM_1 : String
- ITEM_2 : String
- ITEM_3: String
+ con : Connection
+ pst : PreparedStatement
+ rs : ResultSet
+ Orders() : void
+ Orders(String, String, double, String, String, String, String, String, String) : void
+ getID() : String
+ getName() : String
+ getPrice() : double
+ getStatus() : String
+ getEmail() : String
+ getAddress() : String
+ getITEM_1() : String
+ getITEM_2() : String
+ getITEM_3() : String
+ deleteOrder(String, String, String) : boolean
+ updateOrder(String, String, double, String) : boolean
+ resetOrder() : void
+ updateOrder(String, String, String, String, String, String) : boolean
+ createOrder(String, String, String, String, String, String, String, String) : void
DBConnection
+ con : Connection
+ pst : PreparedStatement
+ rs : ResultSet
+connect() : Connection
Inclusion
Inclusion
Inclusion
Inclusion
USER
- User : String
- Pass : String
+ con : Connection
+ pst : PreparedStatement
+ rs : ResultSet
+ getUser() : String
+ getPass() : String
+ checkLogin(String, String) : boolean
+ addRegister(String, String, String) : boolean
Storages
- ID : String
- Name : String
- Price : double
- Quantity : double
- Size : double
+ con : Connection
+ pst : PreparedStatement
+ rs : ResultSet
+ Storages() : void
+ Storages(String, String, double, double, double) :void
+ getID() : String
+ getName() : String
+ getPrice() : double
+ getQuantity() : double
+ getSize() : double
+ addStorage(String, String, String, String, String) : void
+ deleteStorage(String, String, String, String, String) : void
+ updateStorage(String, String, double, double, double) : void

GUI UML


Orders
Inclusion
AddOrder
- A set of jLables, jButtons, jTextFields and a model of table
+ con : DBConnection
+ order : Orders
- jButton3ActionPerformed(evt) : void
- jButton4ActionPerformed(evt) : void
- jButton5ActionPerformed(evt) : void
- AddActionPerformed(evt) : void
- ClearActionPerformed(evt) : void
- showallActionPerformed(evt) : void
- FindActionPerformed(evt) : void
+ AddOrder() : void
+ storageList() : ArrayList<Storage>
+ show_storage() : void
+ findstorageList() : ArrayList<Storages>
+ find_storage() : void
Inclusion
ORDER
- A set of jLables, jButtons, jTextFields and a model of table
+ con : Connection
+ order : Orders
+ orderList() : ArrayList<Orders>
+ ORDER() : void
+ show_order() : void
+ findorderList() : ArrayList<Orders>
+ find_order() : void
+ FindActionPerformed(evt) : void
+ ClrAllActionPerformed(evt) : void
+ Table_OrMouseClicked(evt) : void
+ ViewActionPerformed(evt) : void
+ ShowAllActionPerformed(evt) : void
+ DeleteActionPerformed(evt) : void
+ AddActionPerformed(evt) : void
+ UpdateActionPerformed(evt) : void
+ CalcActionPerformed(evt) : void
+ jButtonActionPerformed(evt) : void
Inclusion
Login
- A set of jLables, jButtons, jTextFields
+ con : DBConnection
+ user : USER
+ en: Encryptor
- RegButActionPerformed(evt) : void
- ForgotButActionPerformed(evt) : void
- LogButActionPerformed(evt) : void
Inclusion
Encryptor
+encryptString(String) : String
Inclusion
USER
Register
+ con : DBConnection
+ user : User
+ en:Encryptor
- A set of jLables, jButtons, jTextField
+ Register() : void
+ RegButActionPerformed(evt) : void
+ ClearActionPerformed(evt) : void
+ jTextField2FocusGained(evt) : void
Inclusion
Inclusion
Inclusion
Inclusion
inclusion
Inclusion
VIEWDETAILS
- A set of jLables, jButtons, jTextFields
+ con : DBConnection
+ order : Orders
+ VIEWDETAILS() : void
+ VIEWDETAILS(ArrayList<Orders>) : void
+ showData(ArrayList<Orders>) : void
- UpdateActionPerformed(evt) : void
Inclusion
DBConnection
Inclusion
History
Inclusion
Inclusion
viewHistory
+ con : DBConnection
+ history : History
- A set of jLables, jButtons, jTextFields and a model of table
+ viewHistory() : void
+ History() : ArrayList<History>
+ show_History() : void
+ jButton2ActionPerformed(evt) : void
+ jButton1ActionPerformed(evt) : void
+ jButton3ActionPerformed(evt) : void
Inclusion
Inclusion
STORAGE
+ con : DBConnection
+ storages : Storages
- A set of jLables, jButtons, jTextFields and model of a table
+ Storage() : void
+ storagesList() : ArrayList<Storages>
+ show_storage() : void
+ findstorageList() : ArrayList<Storages>
+ find_storage() : void
- Table_StoMouseClicked(evt) : void
- updatebutActionPerformed(evt) : void
- addbutActionPerformed(evt) : void
- findbutActionPerformed(evt) : void
- showallActionPerformed(evt) : void
- deletebutActionPerformed(evt) : void
- jButton1ActionPerformed(evt) : void
- jButton2ActionPerformed(evt) : void
Inclusion
sendFeedbacks
+ con : DBConnection
- A set of jLables, jButtons, jTextFields
- jButton1ActionPerformed(evt) : void
- jButton2ActionPerformed(evt) : void
Inclusion
Storages
Menu
- A set of jLables, jButtons, jTextFields
+ con : DBConnection
- storagebuttonActionPerformed(evt) : void
- feedbuttonActionPerformed(evt) : void
- orderbuttonActionPerformed(evt) : void
- historybuttonActionPerformed(evt) : void

Classes Function

Class' name	Function
DBConnection	Get access to the database
Login	Allow users to type in their username and password, make comparisons with existing data in the database, and give them permission to access if they match.
Register	Allow users to type in their username, password and email; do some checkings with the primary key variable and insert to the database if it is valid
Encryptor	Encrypt the registered passwords before inserting them into the database
Menu	Provide the "gate" to access all of the features of the application
STORAGE	Show in a table, all the items that are available. Users are able to edit and update it
Storages	Contain the accessors and mutators functions of the object "storage", each object is an item
ORDER	Show in a table, all the orders that the store has. Users are able to edit and update it
Orders	Contain the accessors and mutators functions of the object "order"
sendFeedback	Allow user to report the problems, and bugs from the program to the developer via email
viewHistory	Show in order in a table, the list of major activities have been done in the applications
AddOrder	Allow user to create new order
VIEWDETAILS	Display detailed information of selected order
History	Contain the accessors and mutators functions of the object "History"

Data Dictionary

TABLE: History			
Column	**Data Type**	**Constraint**	**Description**
Action	String	NONE	recorded acion from the application

TABLE: User_In4			
Column	**Data Type**	**Constraint**	**Description**
USER	String	NOT NULL, UNIQUE	unique user's username
PASS	String	NOT NULL	user's password
EMAIL	String	NOT NULL, UNIQUE	unique user's email

TABLE: storage			
Column	**Data Type**	**Constraint**	**Description**
ID	String	NOT NULL, UNIQUE	unique item's ID
Name	String	NOT NULL	item's name
Price	Double	NOT NULL	item's price
Quantity	Double	NOT NULL	item's quantity available
Size	Double	NOT NULL	item's size (shoes)

TABLE: order1			
Column	**Data Type**	**Constraint**	**Description**
ID	String	NOT NULL, UNIQUE	unique order's ID
Name	String	NOT NULL	customer's name
Price	Double	NOT NULL	committed price between customer and seller
Status	String	NOT NULL	order's status
Email	String	NOT NULL	customer's email
Address	String	NOT NULL	customer's address
ITEM_1	String	NONE	first item's ID
ITEM_2	String	NONE	second item's ID
ITEM_3	String	NONE	third item's ID

Proposed Queries

Query Type	Function
Insert	Insert data to a certain table in the database
Insert or Ignore	Insert data to a certain table if the duplicate variable is not existed
Update	Update the data in a certain table with match given information
Select	Select rows of data that match with the given information
Delete	Delete a row of data from a certain table with match given information

Test Plan

No	Success Criteria being tested	Method of testing	Expected Result
1	The users are able to log in using a personal username and password to access the application	The usernames and passwords filled in the fields are checked with the existed database. If it matches, continues to process	The “Menu” window appears after the correct username and password are filled in
2	The program is able to auto-update the quantity of the corresponding products in the Storage table after an order is created with that products	After an order is made, the application will fetch from the “storage” table the quantity of for each item and update it	The quantity of the chosen item in the “storage” table decreases when an order is made
3	The program can access a local database with multiple tables in order to update and monitor the application	Create “DBConnection” class to connect the database to the application. Testing the functionality of all the queries in the application. Using the workbench DB Browser to monitor the activities in the database	A connection is made between the application and a database. There are queries used to adjust the data in the database
4	The application will have a feature to keep track of the actions that happened in the application, including adding, removing orders or items, creating new accounts, and deleting orders,...	At the end of every successful action, a query is used to add the action to a “History” table in the database.	A table of done activities in the application is displayed with a click of a button
5	The users will be able to use queries to manipulate the database such as adding or removing orders and items, and changing the status of the orders, …	Using queries to make adjustments, to make sure the usage of the “DB Browser” application is used to check whether the elements have been adjusted or not	The adjusted elements appear in the database every adjustment

6	The program is capable of doing simple revenue calculations	After click the button, the application will fetch all the data from the orders table into an ArrayList, from here a loop from the beginning to the end of the list is used to calculate the sum of all the price.	The value of money made appears after a click of a button
7	The program can sort through the database to find a specific or group of orders/items information and display those orders/items	After choosing the criteria, a select query is used with matched criteria. Matched findings are inserted into an ArrayList and the table will fetch the data from that ArrayList	A table with match data is displayed after a click of a button with the criteria of the findings are filled
8	The program will be able to notify users if there are errors, such as blank text fields, wrong passwords,...	If the user forgot to fill the needed fields, a pop-up message will appear telling the user that fields are empty	When there is an error, instead of crashing, an option pane will appear with a message
9	The users can be able to delete all the data at the beginning of the month or whenever they want by clicking a button	When user click the button, a pop-up message appears and ask if the user wants to continue. If yes, a query is used to clear the data	All of the data is cleared when the user clicks the button
10	The users can be able to send feedback to the developers by sending emails to the developers	When users enter the feedback and the user's information, those things will be sent to an email used by the developer	The feedback of users will be sent to a developer email
11	The password inserted into the program's database will be hashed to ensure the security of the application	When a user successfully register a new account, a database workbench (DB browser) will be used and checked if the password is encrypted or not	The newly registered passwords will be encrypted and then insert to database