Lost and Found App: Breaking the Wall of Lost Items on Campus

Dang Quynh Chi

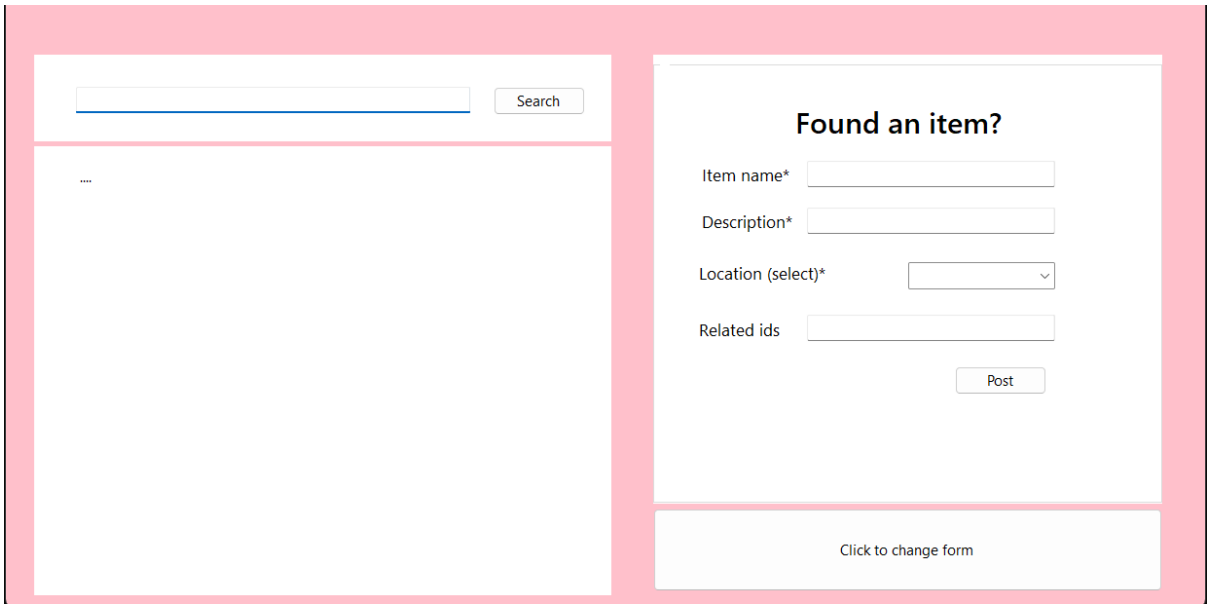
COS20007

11/26/23

Contents

Summary of Program

Every day at your school, valuable items are misplaced - phones, wallets, laptops and more. Without an effective system to record and reunite lost items with their owners, these items often go missing forever.
But now there is a solution – Lost and Found, a centralized lost and found database designed specifically for schools. It allows students and staff to easily log any found items, search for missing belongings, and get credit for returning lost property.
By cutting out disorganized paper records and manual searches, the program brings order and efficiency to the chaotic process of reuniting owners with their lost items. No more rummaging through lost boxes hoping for the best.
Sensitive item details are stored using encryption and salting, so you can feel safe logging expensive devices without fear of theft. User logins are also protected by passwords or accessed by google sign in (fpt mail)



Required Roles

Class: Identifiable object

Responsibility	Notes	Type details
_identifiers	Id of the item	List<string>
IdentifiableObject	(string[] indents) add ids to id list	Constructor
`AddIdentifier`	(string id) add id to be used for Item	String

Class: ProgramObjects

Responsibility	Notes	Type details
Identifiable	(List string) adds ids	constructor
ident		List<string>
AreYou	If contains id, return	bool

Class: Location :ProgramObject

Responsibility	Notes	Type details
_location_key	The key of the location used to identify item location	int
Location	Initialises the _location_key	Constructor (string[] id, string name, string desc, int location_key)
GetKey	Return the key	int
HasKey	Checks if location has key	bool

Class: LocationList

Responsibility	Notes	Type details
LocationList()	Initializes the list of locations	constructor
locations	The list of locations	List<Location>
AddLocation	Add locations into the list	void
GetList	Return the list of locations	List<Location>
Locate	Locates location based on string	Location
HasKey	Checks if any of the locations has key	Location

Abstract Class: Command

Responsibility	Type details	Notes
Execute	`Location location, string text`	
server	`SqlServer`	Initializes has a relationship

CommandFactory

Responsibility	Type details	Notes
makeCommand	Creates command based on input string	Command

Class: LookCommand :Command

Responsibility	Notes	Type details

Execute	Look for item using the sql server method find	Method string

Class: Abstract SendForm

Responsibility	Notes	Type details
SendForm		constructor
Execute	Blueprint execute command for classes	abstract string

Class: Take:SendForm

Responsibility	Notes	Type details
Execute	Take item using the SqlServer method	override string

Interface:FormFactory

Responsibility	Notes	Type details
MakeForm	Make the appropriate command based on string input	SendForm

Class: Poste:SendForm

Responsibility	Notes	Type details
Execute	Post item using the SqlServer method	override string

Interface: ILocationList

Responsibility	Notes	Type details
Locate(string id)		Location
GetList		`List<Location> GetList`
AddLocation		`void`

Class: SqlServer

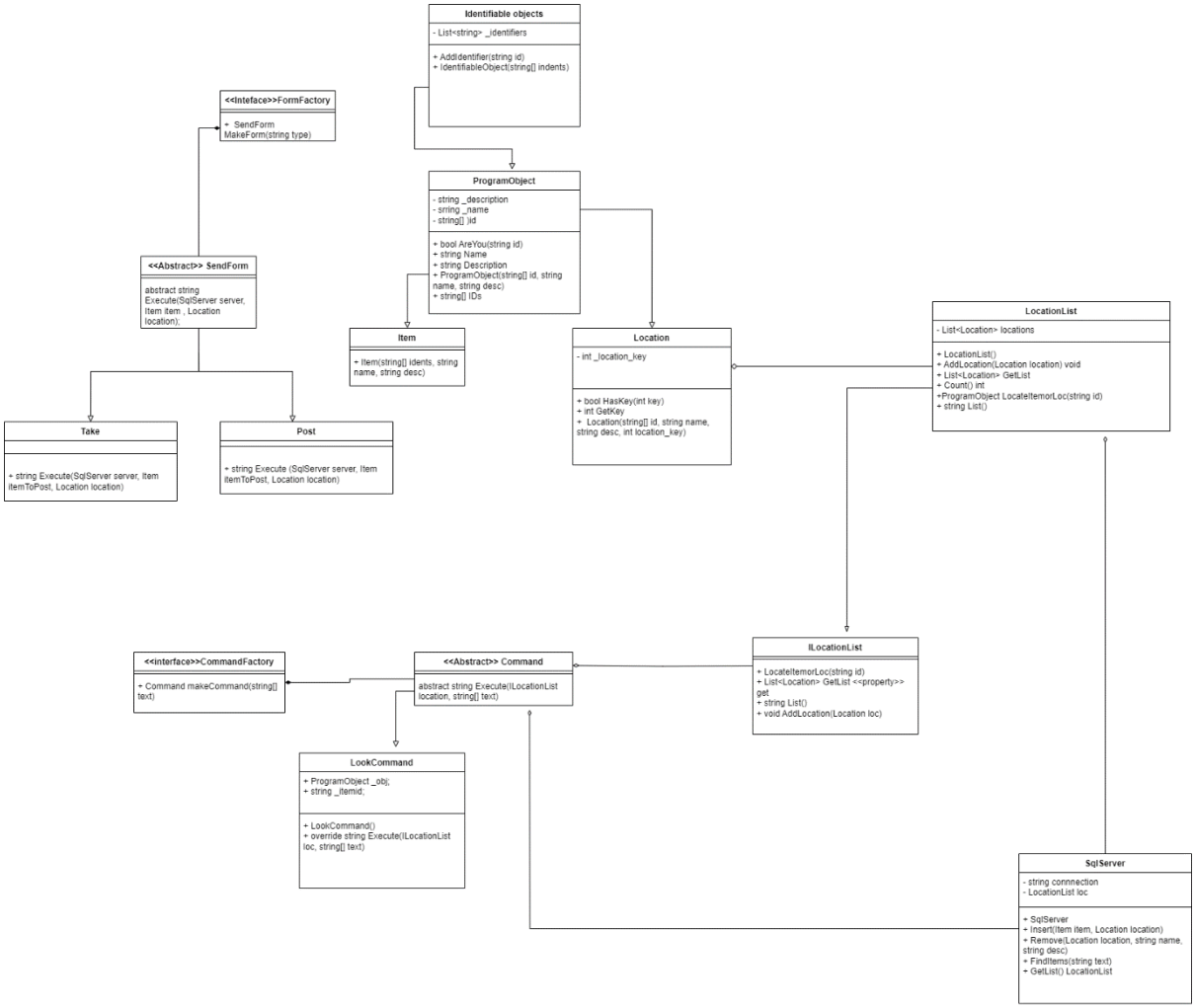
Responsibility	Notes	Type details
connectionString	The default connection string, therefore we would only need to re-establish it once throughout the program. This avoids silly mistakes.	private readonly
list	The list of locations used for searching	LocationList
Insert	Opens the connection and do a query to insert items into the table	string
Remove	Opens the connection and do a query to insert items into the table	string

FindItems	Locates item based on string. This uses the Get list command in the same class to get the location name	List<string>
FindLocation	Mostly for maintenance. I separated into 2 separate classes as they both uses different queries. This helped in preventing any silly mistakes that can shut down the enter finding item process	void
GetList	Gets the data from the table and insert it into the locationlist class. This is the bridge for the drop down box in the Form	LocationList

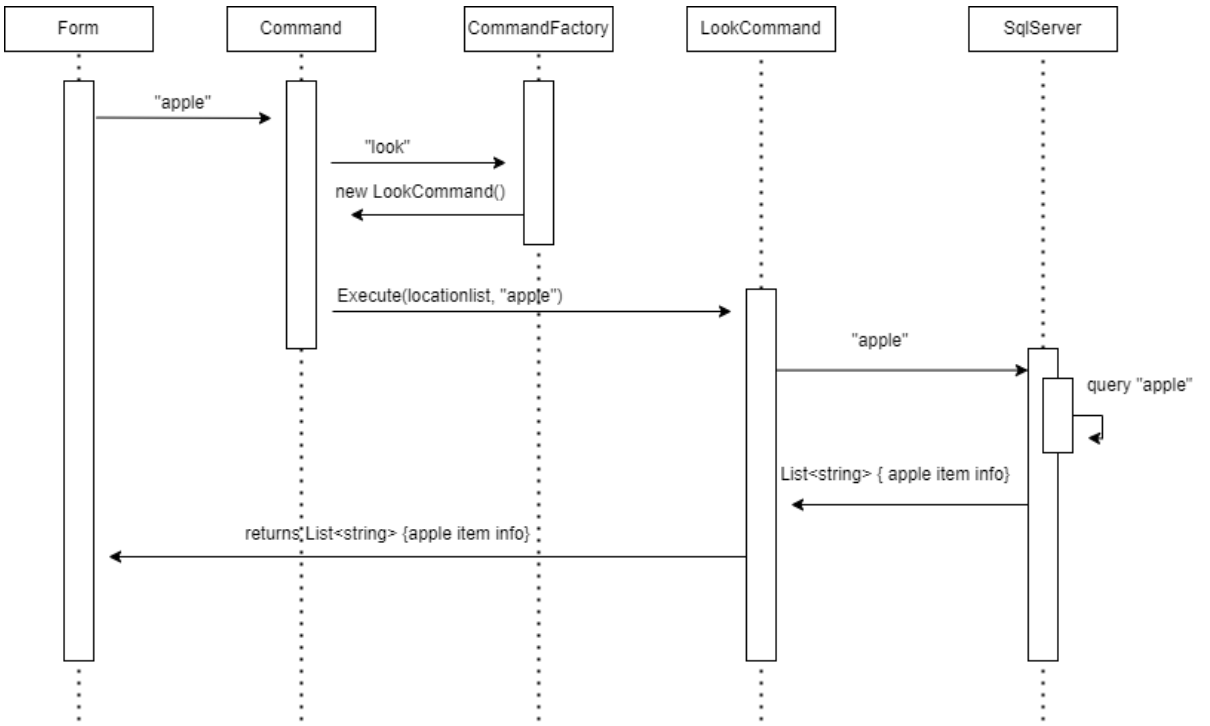
MySqL Table

Responsibility	Notes	Type details
Items	Stores the item data. Is connected to the Location table using the key	
Locations	Stores the location data. Is connected to the Items table using the key	

Class Diagram



Sequence Diagram



Design pattern implementations

Factory pattern

The Factory Method pattern abstracts the decision-making process from the calling class. This has several advantages:

1. **Reuse.** I don't need to re-establish my conditions if I want to instantiate in multiple places. Therefore, the chance of missing one when adding a new class is significantly decreased.

2. **Access** the factory acts as a bridge between the products and the ingredients. In this case it is the information needed to create the commands. By creating a distance between the two, it effectively helps unit testing in the future and de-coupling.

3. **Unit-Testability.** I can write 3 tests for the factory, to make sure it returns the correct types on the correct conditions, then my calling class only needs to be tested to see if it calls the factory and then the required methods on the returned class. It needs to know nothing about the implementation of the factory itself or the concrete classes.

4. **Extensibility**. When someone decides we need to add a new class D to this factory, none of the calling code, neither unit tests or implementation, ever needs to be told. We simply create a new class D and extend our factory method. This is the very definition of Open-Closed Principle.Dependency injection

Dependency injection (Construction Injection)

1. **Highly Extensible Code** The codebase is expected to evolve over time, and one will often (or even constantly) have to fix bugs and defects. However, when using dependency injection because of its loosely coupled nature, one can improve the application quickly with far less effort.
2. **Control** It provides the ability to replace dependencies without changing the class that uses it and the ability to control functionality from a central instead of spreading it throughout your program

After-thoughts:

During the making of this custom program, I've learned a lot. A hard lesson I've learned is to implement and establish the database first. I underestimate how powerful mysql can be, therefore, implementing it in the end, made me realize my program could go server-client completely for data,(only using tables wo creating any objects) therefore diminishing the need for different classes, and focusing on the commands (although it would make the server reliant on the sql server, I feel like it is worth a try). Overall, I would say this helped me learn a lot in terms of OOP and thinking in ways that makes debugging and maintaining the program a lot more convenient.