PROJECT REPORT: COUNTER AND STORAGE SYSTEM

THESIS TYPE: PROJECT REPORT
COURSE NAME: DLBCSPSE01
COURSE OF STUDY: COMPUTER SCIENCE
DATE: 30.10.2024

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

There are many ways in which people enjoy video games, some like role playing games, others shooters, racing etc... But there is a community of people who enjoy a particular style of gameplay called 'Shiny hunting'. The premise is to hunt special creatures, called Pokémon, which have a very small chance of appearing. For example, the video games released in 1999, Pokémon Gold and Silver, in which finding a shiny is 1/8192.

While hunting these special creatures, a lot of individuals like to keep count of how many encounters they have made, and to know just how lucky (or unlucky) they were once they actually do find it. This led to the idea of creating an application that would help individuals keep count and also store all their successful tries.

Naturally, just keeping track of encounters is simple and could be done on a basic calculator, therefore the application should provide more than just a simple increment for each encounter.

1.1 Problem Statement

Since a lot of people do not exclusively play games on their desktop computer, it is essential to have this sort of application be available on mobile devices like phones or tablets.

That is why this application focuses on a mobile first approach, where the application is first developed for phones and then translated into a desktop version. This helps it be more mobile friendly and makes the development smoother and more dependable.

Another problem many people nowadays face is the concern about the security of personal data. Most websites nowadays want to access peoples search histories or other data, which for obvious reasons many dislike.

This is why it was very important for this application to not collect any data from users, furthermore the applications database is strictly stored on the users local browser, meaning that there are no login requirements. This further increases user satisfaction, since there is no need to remember passwords or share their personal information like email addresses to the internet.

1.2 Current State

Currently the project is deployed and already available to users on GitHub using GitHub pages. The main objectives of the website have been met, and it has already helped people with their shiny hunts.

At present, the site works as intended and described in these specifications, with potential updates and improvements in the future.

CHAPTER 2

REQUIREMENTS SPECIFICATION

This chapter will focus on the main objectives of what this application is trying to achieve. Furthermore, it will provide similar applications made in the past and pinpoint its flaws. It also provides functional and non-functional requirements for the project and finishes with a system model and user interfaces.

2.1 Objectives

One of the main objectives of this application is to give users a simple to use, no registration, and no other sort of action like downloading requirements to access the application. A lot of people do not want to go through these processes, even when they are quick and simple, and just want to have the desired product ready to go.

Another objective is to provide a customizable user interface. Since the process of shiny hunting can take a very long time, users will spend a lot of time on the website, meaning they will most likely want to personalize it to some degree to make it more enjoyable.

One of the most important parts however is the functionalities to count and store these encounters.

2.2 Related Work

Since shiny hunting has been around for decades, there have naturally already been programs that do the same thing, however, most of them, including the most popular ones, have some flaws and incompleteness to them.

One of the most important ones is the ability to store completed hunts. Many shiny counters out there only allow you to count your current hunt, but once the hunt is finished and a user wants to go to the next, the progress of the hunt is reset to 0 and the user needs to either write down the count for the previous hunt, or forget it entirely.

A lot of shiny counters are also not visually appealing, meaning they only have a basic increment of numbers and nothing else. To make the application more enjoyable to users, the image of the current hunted creature will be displayed, alongside a customizable background image and wallpaper image for the storage box. This should give users a better view of the app and give them more reasons to use it.

2.3 Functional Requirements

2.3.1 Counter

The most important functionality of the system is naturally the counter itself. It needs to increment the number by 1 when the increment button is clicked. Additionally, there needs to be a decrements button, in case the user miss clicks and wants to go to a lower number. In some cases, the users will want to increment the number by 5 instead of 1, therefore an increment button for +5 should also be available.

2.3.2 Storage Box

After the user finishes with their hunt, there needs to be a button that ends the count. The buttons should provide two possible scenarios. Scenario one is when the user completes the hunt and wants to store it in the box. This should stop the current hunt and store the completed one in the storage below. The second scenario is when the user wants to put their current hunt on hold and start a different one. In this case, the hunt should be stored in a different storage box, where in-progress hunts are stored.

2.4 Nonfunctional Requirements

2.4.1 Performance

While performance in a distributed systems is always desired to be the best it can be, in this application it is not the main priority. The website needs to function correctly and with minimal delay, however it is a simple counter and storage, meaning that it won't require heavy resources to function properly. Therefore, with minimum effort, a desirable performance should be achieved.

2.4.2 Security

Since each user has their own unique storage box, the storage system needs to be secure to avoid mischievous behaviors or accidental user breaches. Since one of the main objectives of this website is simplicity both in usage and in the initial setup, registrations and other sorts of authentications are not desirable. Therefore, all the data will be stored on the user's local browser storage, meaning that there will be no requirements to set up, and each user will have their own personal storage.

2.4.3 Reliability

It is important that this application is reliable and that new versions of the system do not break the current progress of a user. Therefore, the system requires testing and version control, to ensure that the storage system stays consistent and does not corrupt the existing data.

2.4.4 Supported devices

The majority of internet websites nowadays are viewed through phones. However, it is not as simple in this scenario. Most people using this website will be using it at home, most likely while playing on their desk or couch. Therefore, it is not as simple to predict on which kind of device most users will access the website. It is safe to assume though, that the majority of users will be either on their phones or desktop/laptop computers.

2.5 System models

2.5.1 Use Case Model

In this section, the previously discussed specifications are realized with a use case model of the application. The two main actors are the user using the application and the external API. It describes all the available uses of the system and how they interact with each other.

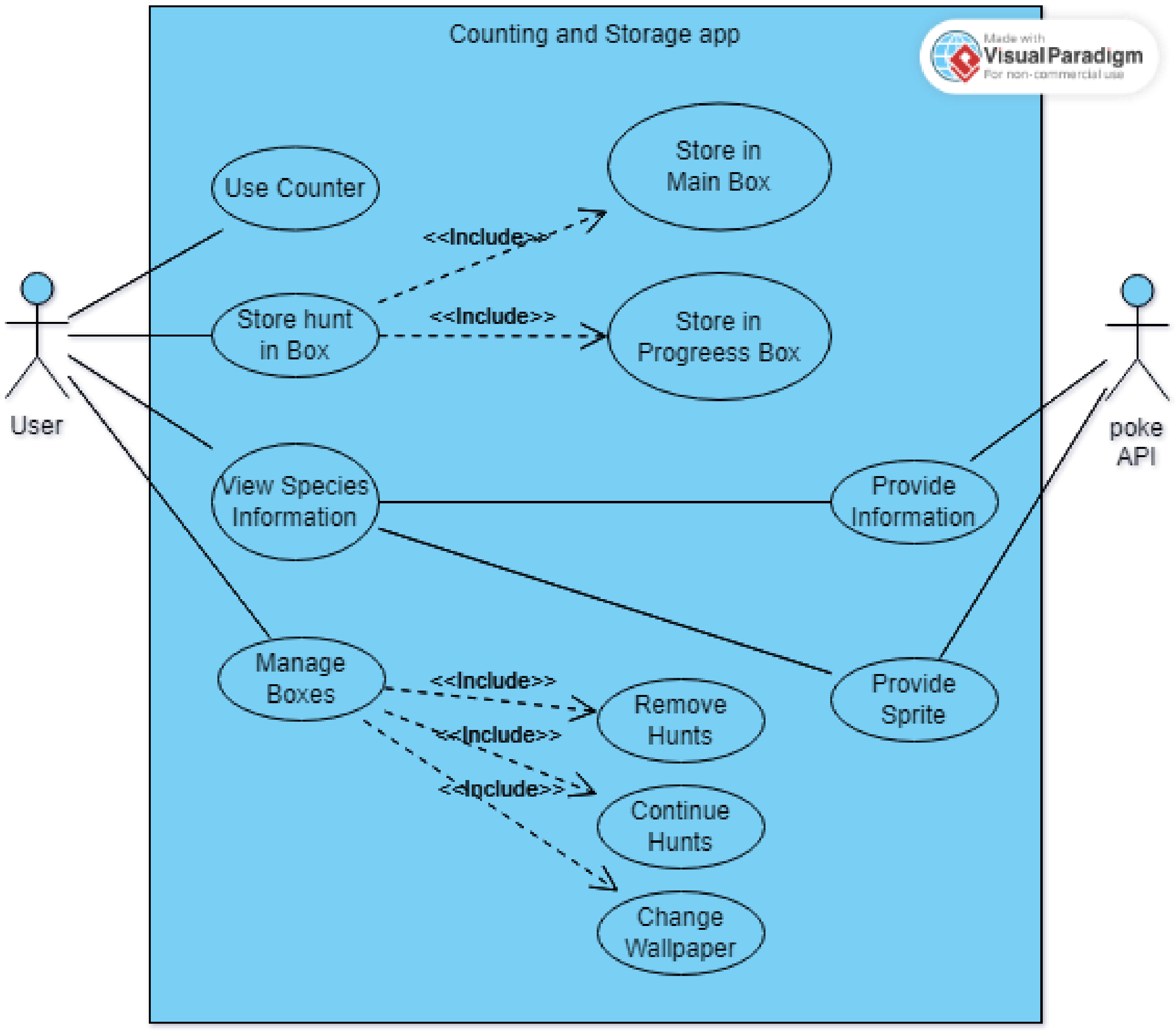


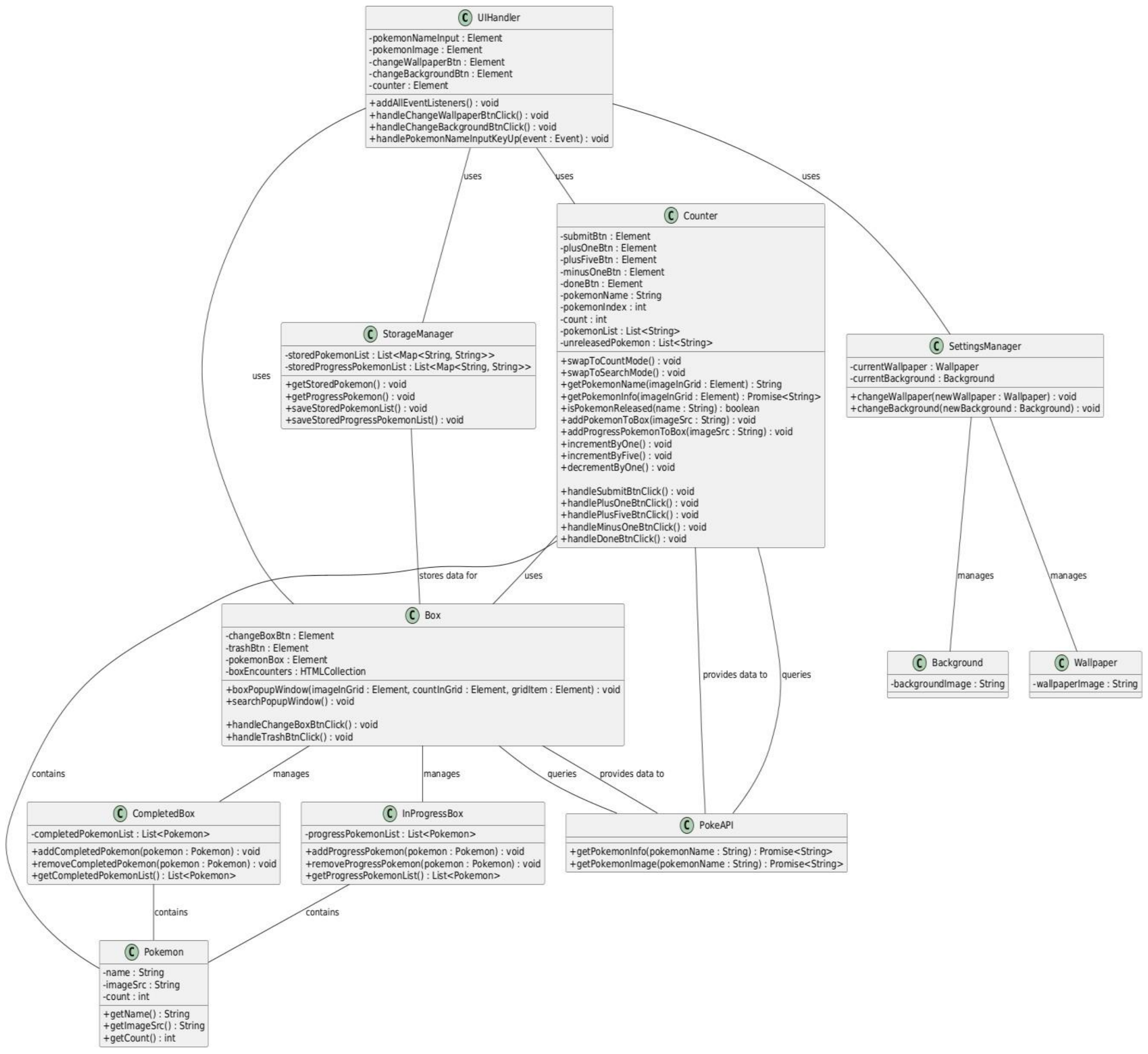
Figure 1

2.5.2 UML Class Diagram

The main two classes, Counter and Box, communicate through the abstract class UIManager, which uses handler and modification functionalities to modify the data of those two classes.

Alongside those classes is also the StorageManager, which gets, loads or removes data directly from the box, or through the UIManager when changing backgrounds or wallpapers.

Finally, there is the PokeAPI class, which gets the data and images from species using an API.



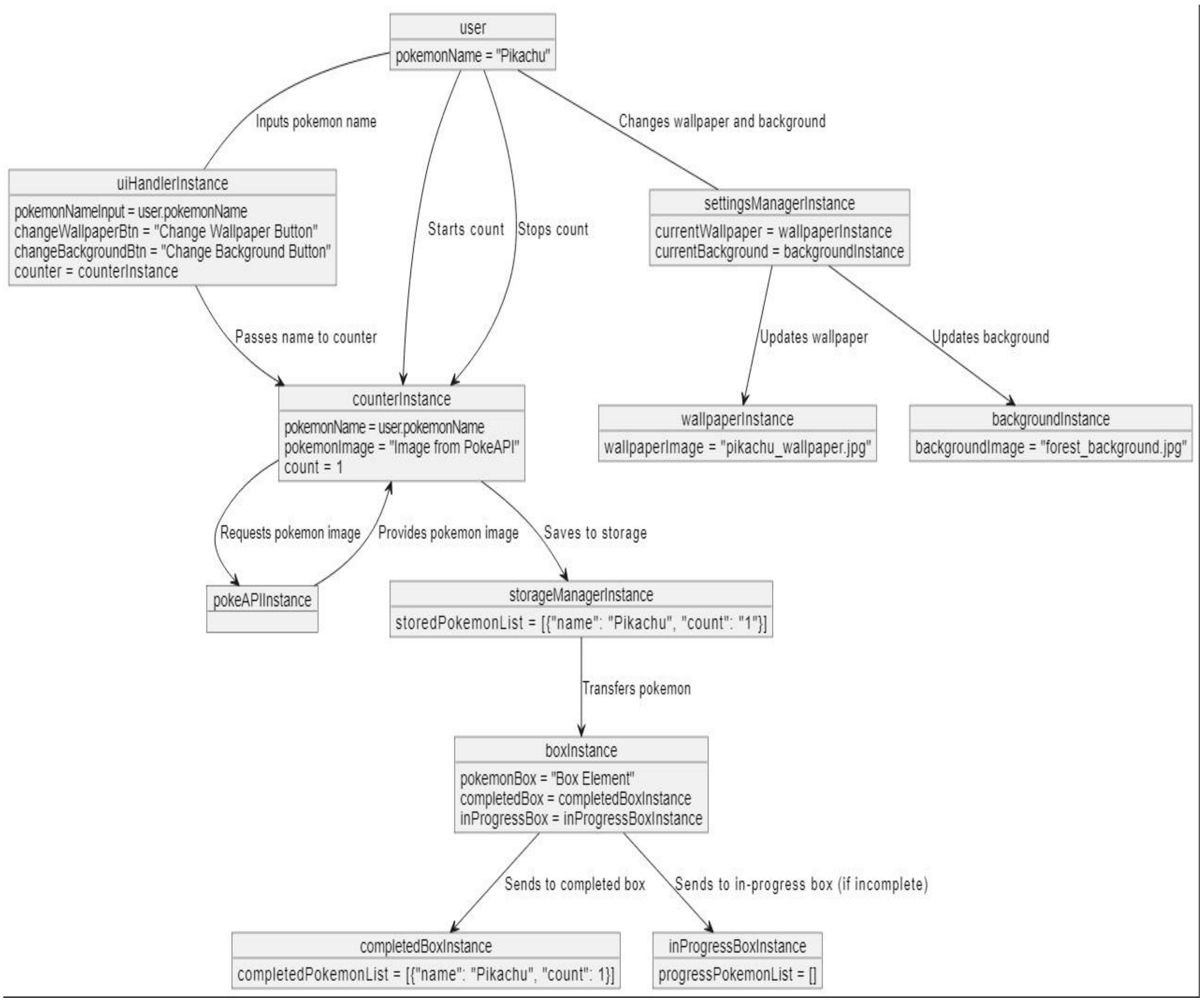
UIHandler
-pokemonNameInput : Element
-pokemonImage : Element
-changeWallpaperBtn : Element
-changeBackgroundBtn : Element
-counter : Element
+addAllEventListeners() : void
+handleChangeWallpaperBtnClick() : void
+handleChangeBackgroundBtnClick() : void
+handlePokemonNameInputKeyUp(event : Event) : void
uses
uses
uses
uses
Counter
-submitBtn : Element
-plusOneBtn : Element
-plusFiveBtn : Element
-minusOneBtn : Element
-doneBtn : Element
-pokemonName : String
-pokemonIndex : int
-count : int
-pokemonList : List<String>
-unreleasedPokemon : List<String>
+swapToCountMode() : void
+swapToSearchMode() : void
+getPokemonName(imageInGrid : Element) : String
+getPokemonInfo(imageInGrid : Element) : Promise<String>
+isPokemonReleased(name : String) : boolean
+addPokemonToBox(imageSrc : String) : void
+addProgressPokemonToBox(imageSrc : String) : void
+incrementByOne() : void
+incrementByFive() : void
+decrementByOne() : void
+handleSubmitBtnClick() : void
+handlePlusOneBtnClick() : void
+handlePlusFiveBtnClick() : void
+handleMinusOneBtnClick() : void
+handleDoneBtnClick() : void
StorageManager
-storedPokemonList : List<Map<String, String>>
-storedProgressPokemonList : List<Map<String, String>>
+getStoredPokemon() : void
+getProgressPokemon() : void
+saveStoredPokemonList() : void
+saveStoredProgressPokemonList() : void
SettingsManager
-currentWallpaper : Wallpaper
-currentBackground : Background
+changeWallpaper(newWallpaper : Wallpaper) : void
+changeBackground(newBackground : Background) : void
stores data for
uses
manages
manages
Box
-changeBoxBtn : Element
-trashBtn : Element
-pokemonBox : Element
-boxEncounters : HTMLCollection
+boxPopupWindow(imageInGrid : Element, countInGrid : Element, gridItem : Element) : void
+searchPopupWindow() : void
+handleChangeBoxBtnClick() : void
+handleTrashBtnClick() : void
provides data to
queries
Background
-backgroundImage : String
Wallpaper
-wallpaperImage : String
contains
manages
manages
queries
provides data to
CompletedBox
-completedPokemonList : List<Pokemon>
+addCompletedPokemon(pokemon : Pokemon) : void
+removeCompletedPokemon(pokemon : Pokemon) : void
+getCompletedPokemonList() : List<Pokemon>
InProgressBox
-progressPokemonList : List<Pokemon>
+addProgressPokemon(pokemon : Pokemon) : void
+removeProgressPokemon(pokemon : Pokemon) : void
+getProgressPokemonList() : List<Pokemon>
PokeAPI
+getPokemonInfo(pokemonName : String) : Promise<String>
+getPokemonImage(pokemonName : String) : Promise<String>
contains
contains
Pokemon
-name : String
-imageSrc : String
-count : int
+getName() : String
+getImageSrc() : String
+getCount() : int

Figure 2

2.5.3 UML Object Diagram

The following UML object diagram shows key object instances in the application and their interactions during a session. The user provides a name, which the uiHandlerInstance uses and passes to the counterinstance.

It then requests the pokeAPIInstance for an image, which is then displayed with the count. The user can adjust the wallpaper and background, and when the user completed or pauses the count, the counterinstance sends the data to storageManagerInstance, which organizes it based on the user's decision.



user
pokemonName = "Pikachu"
Inputs pokemon name
Changes wallpaper and background
uiHandlerInstance
pokemonNameInput = user.pokemonName
changeWallpaperBtn = "Change Wallpaper Button"
changeBackgroundBtn = "Change Background Button"
counter = counterInstance
Starts count
Stops count
settingsManagerInstance
currentWallpaper = wallpaperInstance
currentBackground = backgroundInstance
Passes name to counter
Updates wallpaper
Updates background
counterInstance
pokemonName = user.pokemonName
pokemonImage = "Image from PokeAPI"
count = 1
wallpaperInstance
wallpaperImage = "pikachu_wallpaper.jpg"
backgroundInstance
backgroundImage = "forest_background.jpg"
Requests pokemon image
Provides pokemon image
Saves to storage
pokeAPIInstance
storageManagerInstance
storedPokemonList = [{"name": "Pikachu", "count": "1"}]
Transfers pokemon
boxInstance
pokemonBox = "Box Element"
completedBox = completedBoxInstance
inProgressBox = inProgressBoxInstance
Sends to completed box
Sends to in-progress box (if incomplete)
completedBoxInstance
completedPokemonList = [{"name": "Pikachu", "count": 1}]
inProgressBoxInstance
progressPokemonList = []

Figure 3

2.5.4 Sequence Models

The three main sequence models are presented below. Classes will always interact with each other via the UIHandler, which for clarity purposes, will be presented as an abstract class alongside variable classes.

1. The main functionality of the app, the count, takes the species from the input, stores the data to the storage manager, and begins the count. Every time the count is changed, the counter updates it on the UI and updates the count in the local storage.

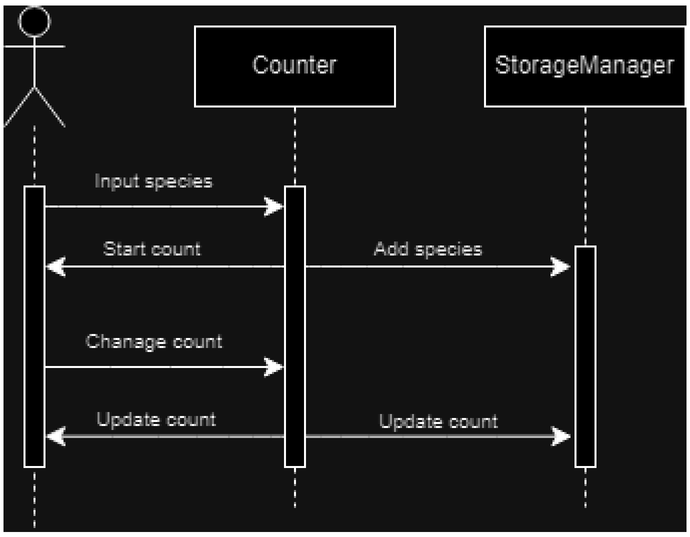


Figure 4

2. Once the user finished the hunt, the user will be asked whether the hunt is finished or not, which will then decide in which box the hunt will be put in. Once the decision is made, the hunt is stored in the correct box, the local storage is updated, and the counter resets for the user to begin the next hunt.

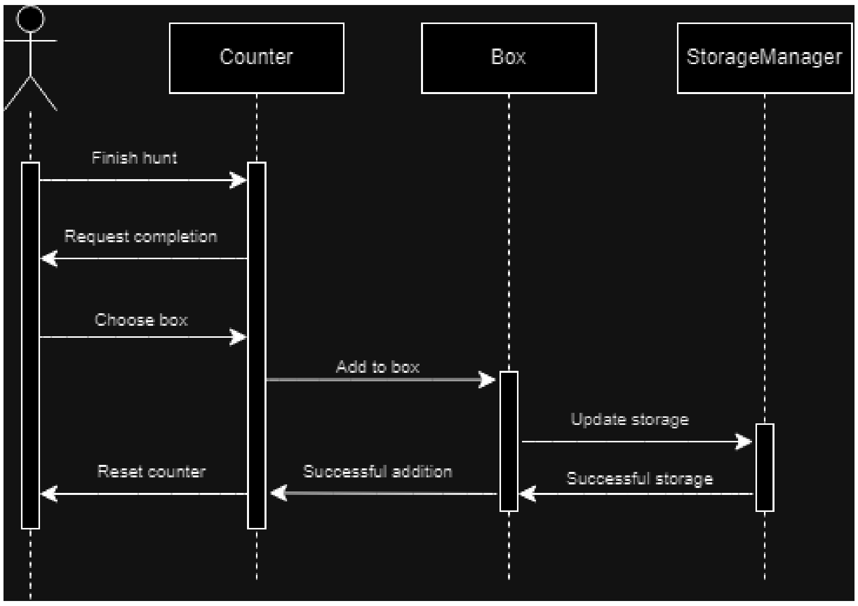


Figure 5

3. To get information of a specific species, the user initiates the sequence by clicking on the desired hunt in the box. A request will then be sent to the API site, if the request is successful, the desired data will be presented to the user. If the data is unable to be retrieved, the user will get an appropriate message of why the information could not be fetched.

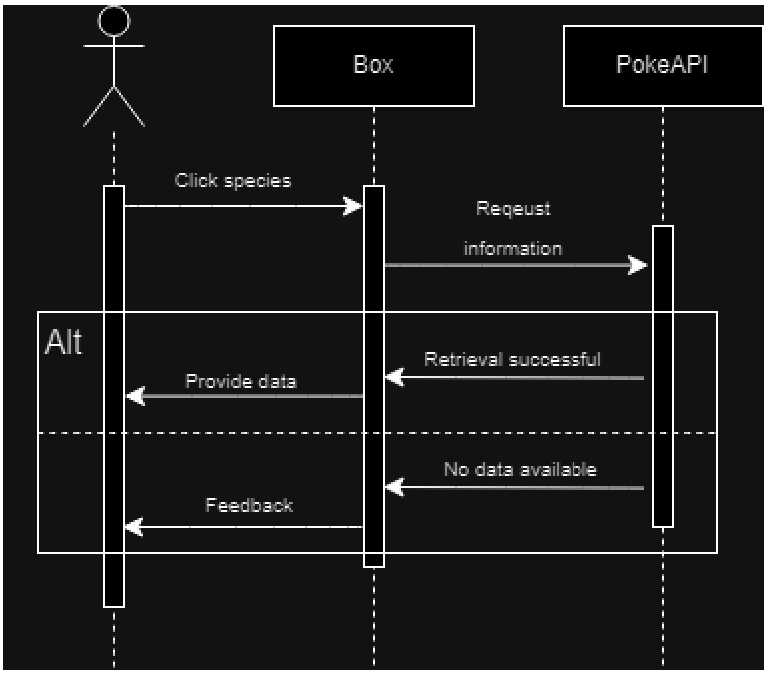


Figure 6

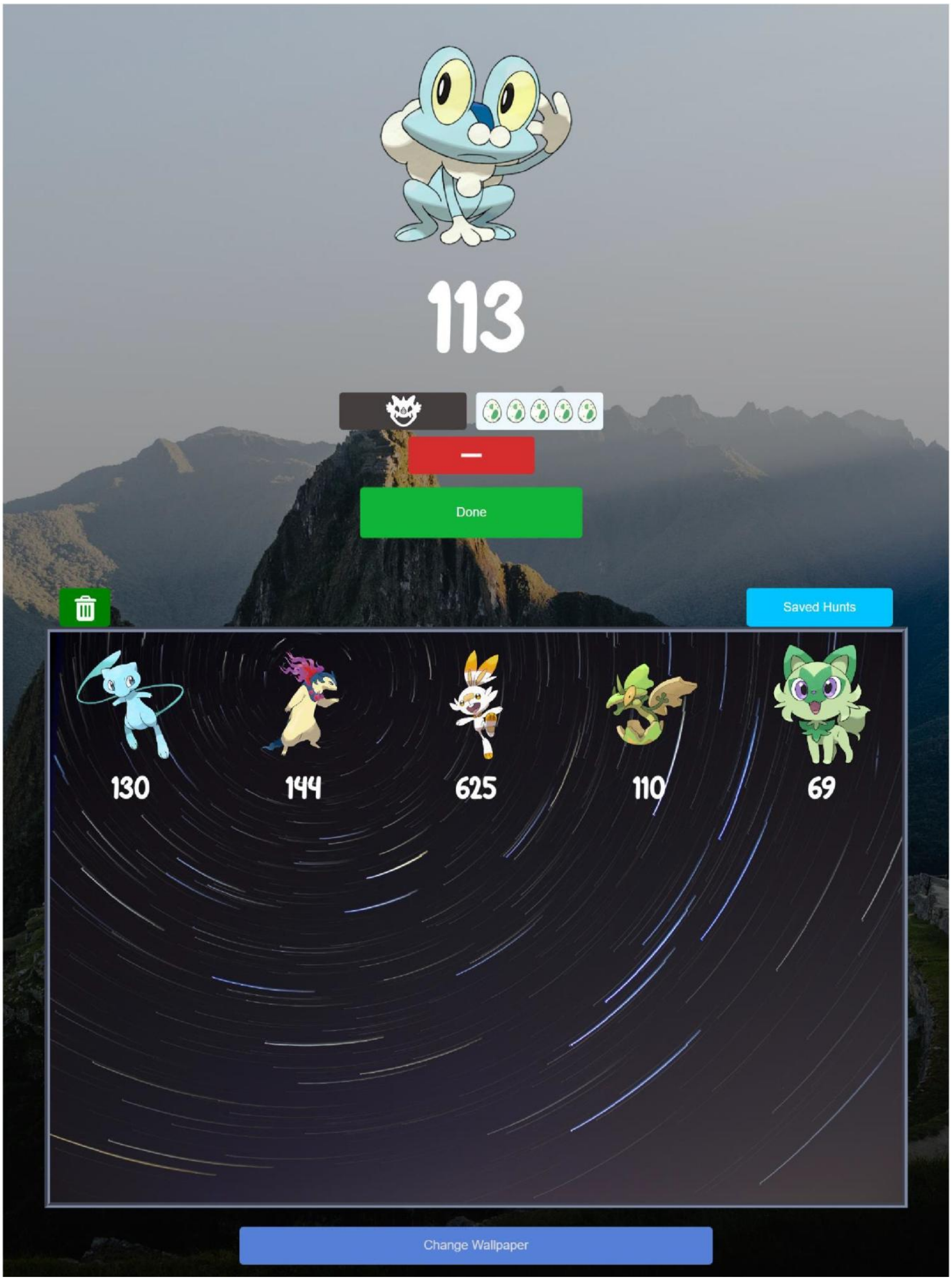
2.6 User Interface

The single page UI of the application is presented below.

At the top is the picture of the current hunt, and below it is the current count. Both the picture and the count take up the majority of the page, as it is the main and most important part of the application.

Going down a bit more are the buttons for changing the count number, the first one on the left is for adding one, and the second is for adding five. The red button below reduces the count by one, and the done button creates a pop-up which asks whether or not the hunt is finished or not. Then there is the main storage box alongside the buttons to change between boxes and to delete desired hunts. The pictures, alongside the count, are presented in each individual hunt in the box using a grid system.

Finally, there is the option to change the wallpaper and background of the application to give individuals room for personalization.



Pokemon images are trademarks of Nintendo©

Figure 7

CHAPTER 3

SYSTEM DESIGN

In this chapter the outline of the design framework of this application will be presented, alongside the system's main goals, essential technologies, and key subsystems. Core subsystems will be decomposed to manage user interactions, data handling, species information, and counting. Finally, the chapter detail how these subsystems are mapped in the hardware and software, optimizing its performance and ensuring good compatibility with devices of clients.

3.1 Design Goals and Enabling Technology

First, an important fact to keep in mind for the design is that the users will spend the majority of their time in the counting section, and only a small portion in the box section. Therefore, the most important design goal is to keep the main portion of the website, the top part of the application, the first and most major part.

Second, the buttons and user input boxes should be large, visible, and with clear intentions. Since the application is not meant to be complex and should be understood by anyone accessing the website, there should be as little confusion as possible for new users. For this to happen, there should be as little buttons as possible, and they should only do one or two things at most. They also need to be distinguishable from each other and also visually appealing.

And finally, the customization of the website should be an option at all times, but it should not interfere with the main application. Meaning that the main functions should be the first thing a user is greeted with when accessing the website, and if they are interested in using more functions, the options would be available outside the main objectives.

The website uses the main web application technologies of HTML/CSS/JavaScript. These will be used to create the user interface, system and function logic, and also the specific designs.

The data will be stored in local storage, which will be accessed and read from with the help of JavaScript. The data such as current hunt species, counts, and completed/saved hunts will all be stored on the users web browser of choice.

3.2 Subsystem decomposition

The main subsystems are the user interface, data management, counting logic, box management logic, species search, and access to APIs.

3.2.1 User Interface

The user interface manages all interactions between the user and the application. This includes the display of the current hunt, current count, and all the completed hunts with their respectful counts.

It also includes the option to change backgrounds and wallpapers to the users desire, alongside useful information on any specific species that the user might be interested in.

3.2.2 Data Management

The data management handles both storage systems and any completed or ongoing counts. Once the application is loaded, the local storage presents all the relevant data and information from the last session, and once the sessions is ended, it stores the data for the next one. There is no need for manual saves, and the deletion is also straightforward, since local storage can be modified at will.

3.2.3 Species Search and API Access

The search species logic uses the pokeAPI to get the desired species information and picture. The API stores species by their respected number, so typing in a random number, for example 292, would get you the species that is assigned to that number. However, most people do not know the exact number their hunt is assigned to, therefore there is also an option to write the species name and get the correct result. The logic behind it is that if the application detects an input of letters instead of numbers, it searches a designated Json file for that species and returns the index of the location in the list.

3.2.4 Counting Logic

The counting logic implements incrementing and reduction of the current count. The main two functionalities are addition and subtraction by one from the main count. Adding for when the next encounter happens, and subtraction if there was a mistake in the count and the count is too high. Additionally, there is a plus five button. This serves for the special occasions when five encounters happen simultaneously, or if the user wishes to get to a higher number and the incrementation by one is too slow.

3.2.5 Box Management Logic

The box management logic comes in two states, the completed box and in progress box. Once a hunt has been finished or the user wants to move on to the next one, they get two options. The first option is to deem the hunt completed and save it in the completed box. This will show the species picture, alongside the final count number below it. Once clicked, the information of that specific species will appear and the option to delete that hunt.

On the other hand, the user might wish to continue the hunt at a later point and start something else. In that case, the option to save it to the in progress box will appear. A hunt in the in this box will also have its picture and count, but clicking on it will give an option to continue the hunt instead of showing the species information. They can also be deleted at will, just like the completed ones.

3.3 Hardware/Software Mapping

The application itself is designed in a way to be able to operate as a lightweight webpage, so that it can be used across various devices and also function on older slightly outdated machines. The following sections show the key hardware and software components needed for the system.

3.3.1 Client Side Requirements

Since the application itself runs entirely on a web browser, it requires minimal client side hardware resources. It should at least have the following:

1. **Processor**: A basic processor is enough for running this application smoothly, because most of the processing is offloaded to JavaScript functions in the browser.
2. **Memory**: Since the website is much more simplistic than an average website, the memory of a device should be sufficient if it is able to run different websites in general.
3. **Storage**: The application uses local storage in the browser, and since local storage is limited to a very small amount per domain, a minimal amount of storage will be sufficient to store the required data.

3.3.2 Software Requirements

1. **Web Browser**: The application supports modern HTML5 compatible browsers such as Chrome, Safari, Mozilla Firefox,... As it uses standard web technologies HTML/CSS/JS.
2. **JavaScript**: All interactive components of the website such as counting, species search, API access,... Are implemented using JavaScript. This includes data retrieval and storage.
3. **pokeAPI**: The application relies on external data access. The API provides images and information which the application retrieves. Therefore an internet access is required to get the necessary data.

3.3.3 Advantages Of The Mapping Approach and Deployment Diagram

The Hardware/Software mapping allows the application to run much easier on different systems, because the responsiveness requires minimal server side resources. Users benefit from faster processing speeds and an available offline access to the data, making it more flexible for them.

It also minimizes the dependency on external servers (except the API), making it more flexible for users who potentially live in environments with limited or inconsistent internet connectivity.

Below, the deployment diagram is presented which shows the relationship between the client device and the web browser, along with the external pokeAPI server.

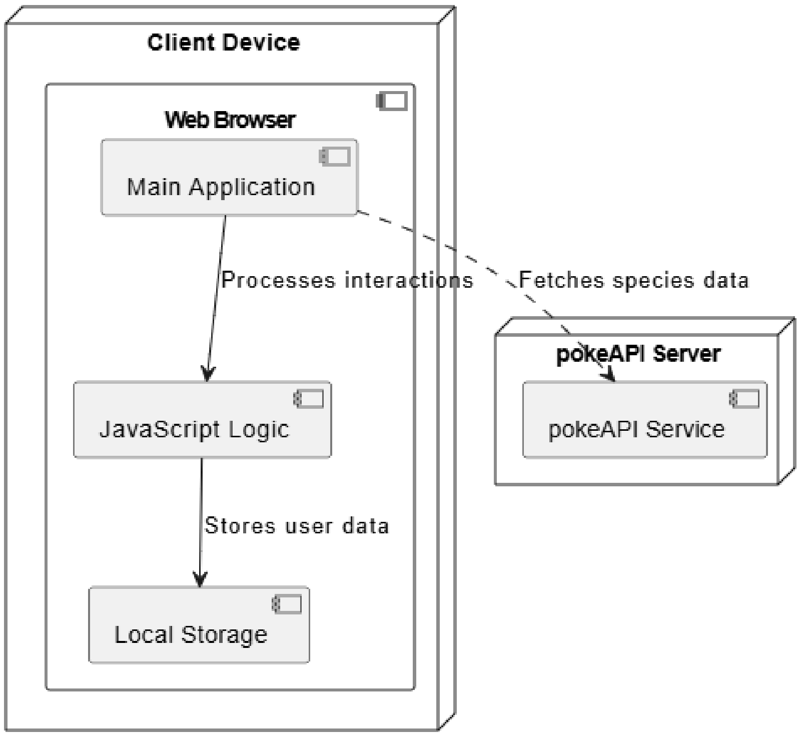

Client Device
Web Browser
Main Application
Processes interactions
Fetches species data
JavaScript Logic
Stores user data
Local Storage
pokeAPI Server
pokeAPI Service

Figure 8

CHAPTER 4

OBJECT DESIGN

In this section, the focus will be to take the previously explained subsystems and classes and define them in greater detail with methods and attributes.

4.1 Package A: User Interface Package

The user interface package includes all classes that are involved in managing user interactions with the application. Below the objects responsible for displaying information and collection input will be defined.

1. **Classes:**

   MainScreen: Handles the layout of the main screen, displays the primary hunt, boxes, and count features.

   Attributes: pokemonBox, boxEncounters, pokemonImage, count, backgroundImage

   Methods: getStoredPokemon(), changeBackground()

   Buttons: Every button in the app

   Attributes: submitButton, plusOneButton, plusFiveButton, minusOneButton, doneButton, trashButton, changeBoxButton, changeWallpaperButton

   Methods: handlers for every button

   InputFields: Includes all input fields on the interface.

   Attributes: pokemonNameInput, placeholderValue

   Methods: getInput(), clearInput()

2. **Diagram:**

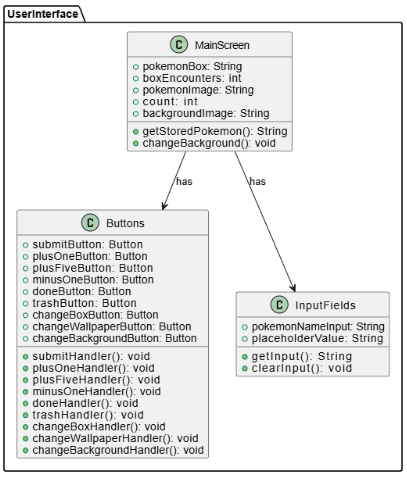


Figure 9

4.2 Package B: Data Management Package

The data management package loads, saves, and deletes data in the local storage. This package will ensure the data is safely stored and track the users progress while counting and storing entries.

1. **Classes:**

   DataManager: Manages the data operations such as saving and loading counts from hunts in local storage.

      Attributes: currentHunt, completedHunts, localStorage

      Methods: loadData(), saveData(), deleteHuntData()

   HuntData: Represents data for individual hunts and if they are completed or in progress.

      Attributes: species, count, status

      Methods: saveHunt(), updateCount(), getHunt()

2. **Diagram:**

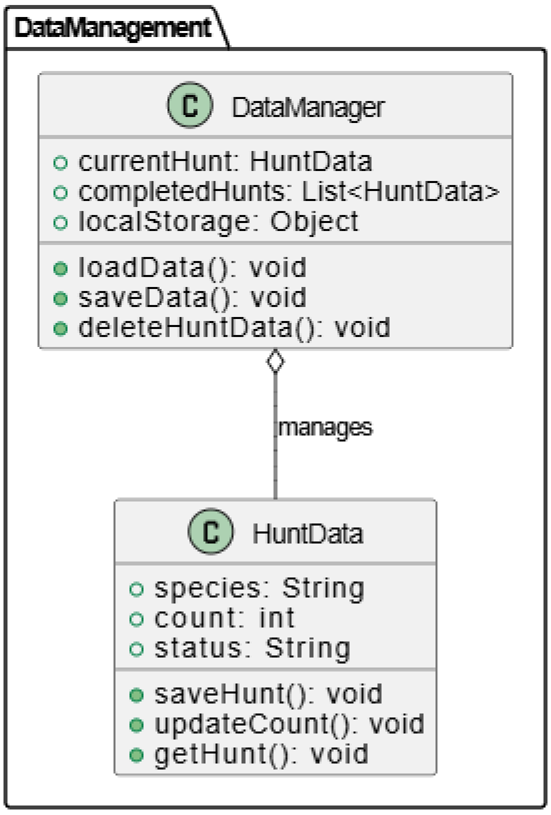


Figure 10

4.3 Package C: Species Search and API Access Package

This package manages external data access mainly from the interactions with the pokeAPI for species information retrieval.

1. **Classes:**

   SpeciesSearch: Manages the species lookup logic, determining whether a number or name is entered and retrieves the correct data.

   Attributes: searchTerm, resultData

   Methods: searchByNumber(), searchByName()

   APIHandler: Manages API requests to pokeAPI

   Attributes: apiEndpoint, requestData

   Methods: getImageSource(), getDataSource()

2. **Diagram:**

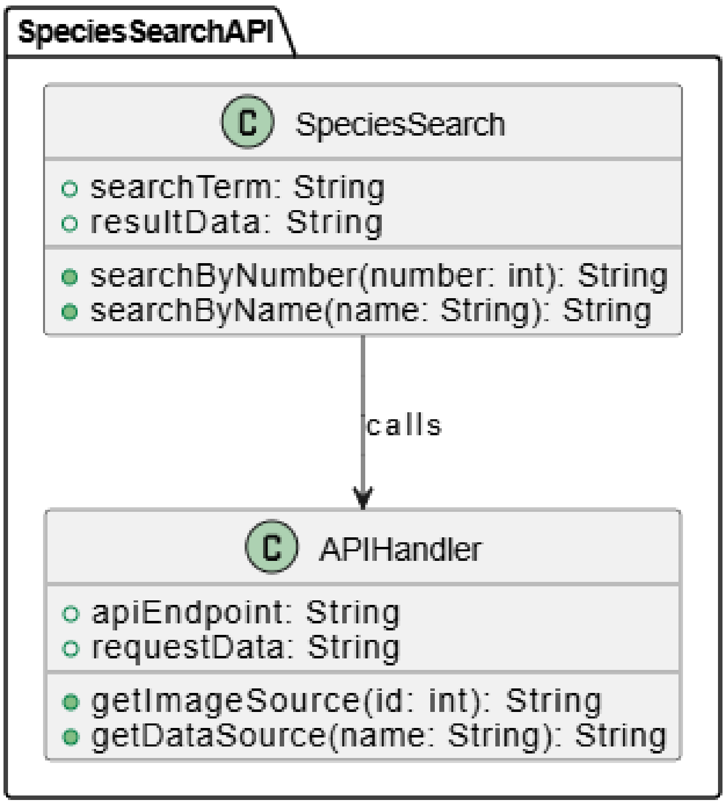

SpeciesSearchAPI
C SpeciesSearch
searchTerm: String
resultData: String
searchByNumber(number: int): String
searchByName(name: String): String
calls
C APIHandler
apiEndpoint: String
requestData: String
getImageSource(id: int): String
getDataSource(name: String): String

Figure 11

4.4 Package D: Counting Logic Package

This package contains the core functionality for counting interactions. It includes classes that allow the user to increase, decrease, or reset the count.

1. **Classes:**

   Counter: Manages the count for a specific hunt.

   Attributes: currentCount, boxCount

   Methods: increment(), decrement(), incrementbyFive(), resetCount()

   BoxCount: Manages the counts from hunts inside boxes.

   Attributes: counter

   Methods: updateCount()

2. **Diagrams:**

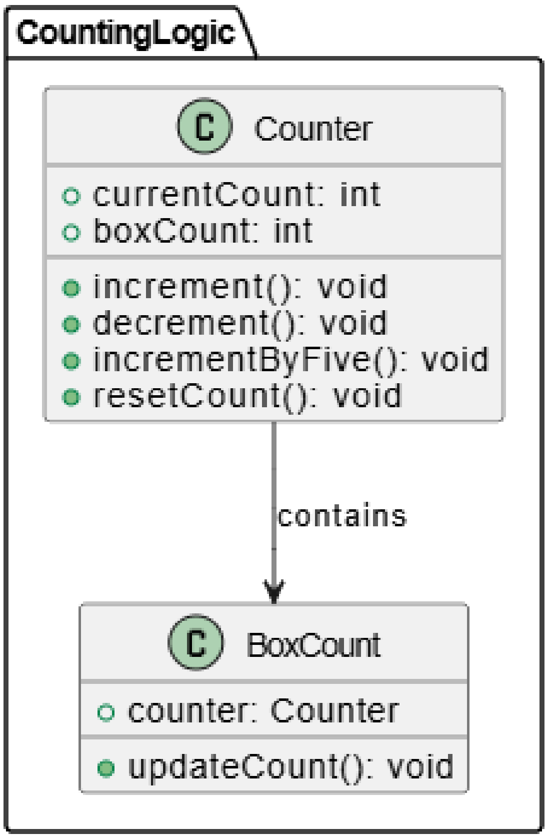


Figure 12

4.5 Package E: Box Management Package

The box management package deals with organization for hunts into "completed " and "in-progress " categories.

1. **Classes:**

   BoxManager: Handles adding hunts to either the completed or the in-progress boxes.

   Attributes: completedBox, in ProgressBox

   Methods: addToCompleted(), addToProgress(), removeHunt()

   HuntBox: Represents each individual box for hunts

   Attributes: boxType, hunts

   Methods: displayHuntDetails(), toggleHuntStatus()

2. **Diagram:**

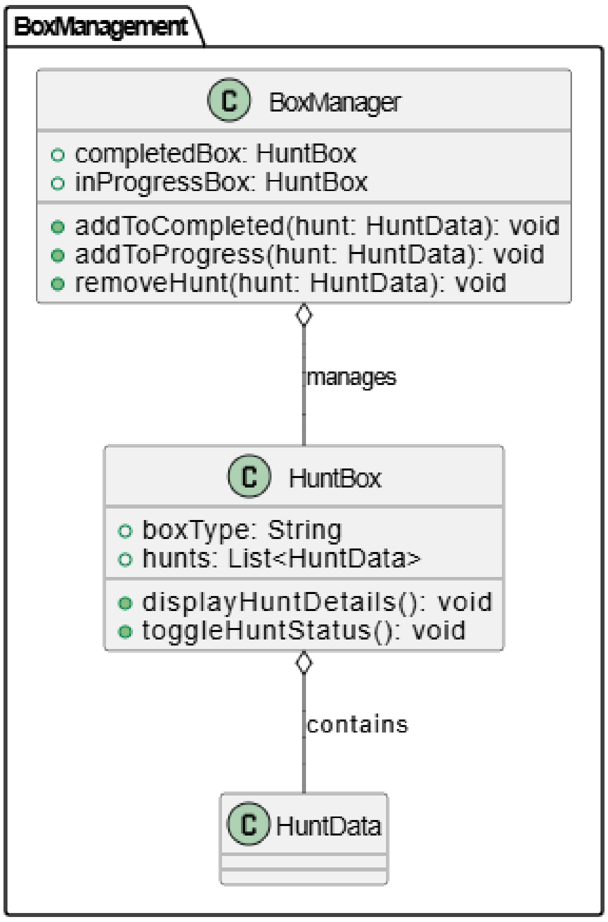


Figure 13

CHAPTER 5

CONCLUSION AND EVALUATION

5.1 Application Repository and Links

This app has been deployed as an open repository on GitHub under the following link: https://github.com/CloudDCrow/shiny-pokemon-tracker-and-counter. The whole code is available to view alongside a readme with the complete instructions.

To visit the website directly the following link is available: https://clouddcrow.github.io/shiny-pokemon-tracker-and-counter/.

5.2 Evaluation of Goals and Methods

The user interface works well on many different devices, is readily customizable, and is simplistic enough so that even people with little to no experience with websites can easily maneuver through it. The API chosen is also very reliable and gets frequent updates when new data is released. It also benefits from being completely open source and allowing contributions.

The downsides of the website, and potential improvement, lie in performance. The page requires large pictures for the background, box wallpaper, and all the sprites, meaning that it takes longer to load compared to an average website. Furthermore, some information gathered from the API does not display correctly, meaning that manual correction will be needed to fix and maintain this issue.

The final lines of code in the integrated development environment is currently at 1237 lines of code.

5.3 Conclusion

There are many options to improve and add functionality to the system, and with it being open source, there might be interested developers in the future that will partake in this project and make it even better than it currently stands.

In conclusion, the system functions and does what it was set to do in the planning process and more. There have also been many people who have left nice messages about the website, and knowing that it has helped some strangers around the world makes this website a success in my eyes.

LIST OF FIGURES