YourBestPet

Additional documentation

NESCOL |30082939

Graded unit 2 project

Radosław Rezler

2023

Table of content:

[Introduction 2](#_Toc133917956)

[Developer manual 2](#_Toc133917957)

[read\_me 2](#_Toc133917958)

[YourBestPet - consumer version 2](#_Toc133917959)

[YourBestPet - developer version 3](#_Toc133917960)

[IDE and extended libraries installation 3](#_Toc133917961)

[Updated UML diagrams 5](#_Toc133917962)

[Sequence diagrams 5](#_Toc133917963)

[Petbook 5](#_Toc133917964)

[Dog finder 5](#_Toc133917965)

[Cat finder 6](#_Toc133917966)

[Pet finder 6](#_Toc133917967)

[Tips 7](#_Toc133917968)

[Use case diagram 7](#_Toc133917969)

[Activity diagram 8](#_Toc133917970)

[New Logo 8](#_Toc133917971)

[Main 8](#_Toc133917972)

[Icon 8](#_Toc133917973)

[Client requirements checklist 8](#_Toc133917974)

[Coding of the problem domain 8](#_Toc133917975)

[Does the code match the design? 8](#_Toc133917976)

[Does the code make appropriate use of inheritance and/or associations? 9](#_Toc133917977)

[Does the code make appropriate use of scope operators? 9](#_Toc133917978)

[Does the code make appropriate use of setters and getters and/or properties as appropriate? 9](#_Toc133917979)

[Does the code make appropriate use of constructors? 9](#_Toc133917980)

[Are the methods and attributes coded correctly 9](#_Toc133917981)

[Coding of the UI domain 9](#_Toc133917982)

[Does the solution match the design? 9](#_Toc133917983)

[Appropriate use of UI controls to ensure that the data entered is valid (eg validation routines and/or error prevention). 9](#_Toc133917984)

[Coding of appropriate events. 9](#_Toc133917985)

[Does the code link appropriately to the business model? 9](#_Toc133917986)

[Use of unfamiliar libraries and/or constructs 10](#_Toc133917987)

[Error handling 10](#_Toc133917988)

[Internal documentation 11](#_Toc133917989)

[Testing strategies 12](#_Toc133917990)

[Test objectives 12](#_Toc133917991)

[Test environment 12](#_Toc133917992)

[Test strategy 12](#_Toc133917993)

[Test Plan/Test Log 12](#_Toc133917994)

# Introduction

This document is an integral part of the second phase of the Graded Unit - developing phase. It contains a detailed description of usage, along with descriptions of individual folders/subfolders, updated charts/logo of the final version of the product, a summary of all client requirements and their implementation, as well as all testing strategies used to quality control.

Please note that this is one of two documents delivered in this project phase. As this is purely technical documentation intended exclusively for the client, it has been separated from the User Manual, which is installed with the consumer version of the product and also added to developer version of the product.

Additionally, both documents are part of the developing phase. This means that further documentation (evaluation report) will be added in later stages of the project.

# Developer manual

This instruction assumes that the Graded “Unit - Radoslaw Rezler - 30082939.zip” folder has been correctly unpacked.

It contains two separate directories “YourBestPet - consumer version” and “YourBestPet - developer version”, as well as a text file named “read\_me”.

## read\_me

The “read\_me” file is a brief description of how to find the extended technical documentation, which is the document you are currently reading. It is crucial to easily obtain all the necessary information regarding the folder structure and instructions for handling different requirements.

## YourBestPet - consumer version

First, let's focus on the consumer version folder. This folder contains an installation file “YourBestPet – Install”, which should be provided to the final users of the product. Clicking on it will open the windows installation wizard, which will install the program and create a directory named “YourBestPet” in C:\Program Files\. There, all the necessary files will be stored for the application to function smoothly, and a backup version of the “User Manual”. After installation, a shortcut to the “YourBestPet.exe” program and “YourBestPet - User Manual.docx” will appear on the desktop. This is a traditional method for installing programs on the Windows operating system, which does not require any technical skills from the user and minimizes any potential complications/errors.

Additionally, in the “YourBestPet - consumer version” folder, there is a directory named “YourBestPet”. This folder was created in case of any issues with the installation of the consumer version of the program (e.g., if there are no permissions to install in the C:/Program Files/ directory). It is the final folder that would appear in the above-mentioned path after the installation from the installer file. It reflects how the directory will look on the client's computer, showing which files will not be present in this product version (like all the technical documentation files and .py versions of the program, enabling code editing).

This folder is not necessary to install the consumer version of the program and was included only in case of installation problems. It also does not contain an additional file "Uninstall.exe" which will appear after installation, as it would not work in this folder (without installation).

## YourBestPet - developer version

Let's now open the “YourBestPet - developer version” folder. It contains all the necessary files for the program to run, along with the extended technical documentation (i.e., this document) and additional program files with the .py extension containing the source code of the program and allowing for editing.

Subfolder “Documentation Part 1” may be outdated, as it is the Planning phase documentation.

Subfolder “Documentation Part 2” contains updated UML diagrams in two different formats, “Radoslaw Rezler - Graded Unit 2023 - Additional documentation” which is this document and “YourBestPet – user manual”.

Please make sure to check and read both documents from “Documentation Part 2” as they are a part of the developing phase of the project.

Another important file is “YourBestPet.exe” and “YourBestPet.py”. First one is executable file that contains the main program (not need to install), second one is a developer version, so the whole python code. Both are in the main “YourBestPet – developer version” folder.

To open “YourBestPet.py”, some integrated development environments (IDEs) will be needed. Additionally, in order to run a program with the .py extension, it will be necessary to install additional libraries: tkinter and Pillow.

In the “YourBestPet – developer version”, there is one more file “YourBestPet – console version.py”. This file has been left in the folder for demonstration purposes only - to show the process of creating the final product. It is not a component of the project deliverable, and some of its algorithms may work differently than the final version. Nonetheless, it is the first completed beta version, which allowed assessing the amount of work needed to complete the final version of the product, identifying preliminary potential problems that may be encountered, and evaluating the sense of the final operation of all functions. This version does not require the import of additional libraries to function properly. However, some sort of IDE is required to open this file.

Please note that the program will only work properly if the YourBestPet.exe file used to open it is located in the folder with the other files needed to open the program. In the case of the program being opened directly from the emergency folder YourBestPet (located in the YourBestPet - consumer version directory) or from the YourBestPet - developer version folder, please do not move the YourBestPet.exe file to any other location. However, you can create a shortcut to this file in any other location. This is what the file installer included in the consumer version does - it creates the main program in C:/Program Files/YourBestPet and then creates a shortcut to YourBestPet.exe on the desktop, allowing the program to run correctly.

## IDE and extended libraries installation

There are many integrated development environments available. The program used to create the project was PyCharm Community Edition. It is an excellent tool for working on advanced projects. Its advantages include automatic project folder creation, automatic library downloading, error hints, and the ability to integrate with GitHub Co-pilot. All of these options speed up the project work and improve its readability. If the user has experience with this program or another advanced code editor that uses multiple colours to highlight different elements of the code and is familiar with downloading additional libraries (such as tkinter and Pillow), it is recommended to use these programs. This will greatly improve the readability of the code and facilitate its configuration.

The PyCharm program is available for free at: https://www.jetbrains.com/pycharm/download/#section=windows

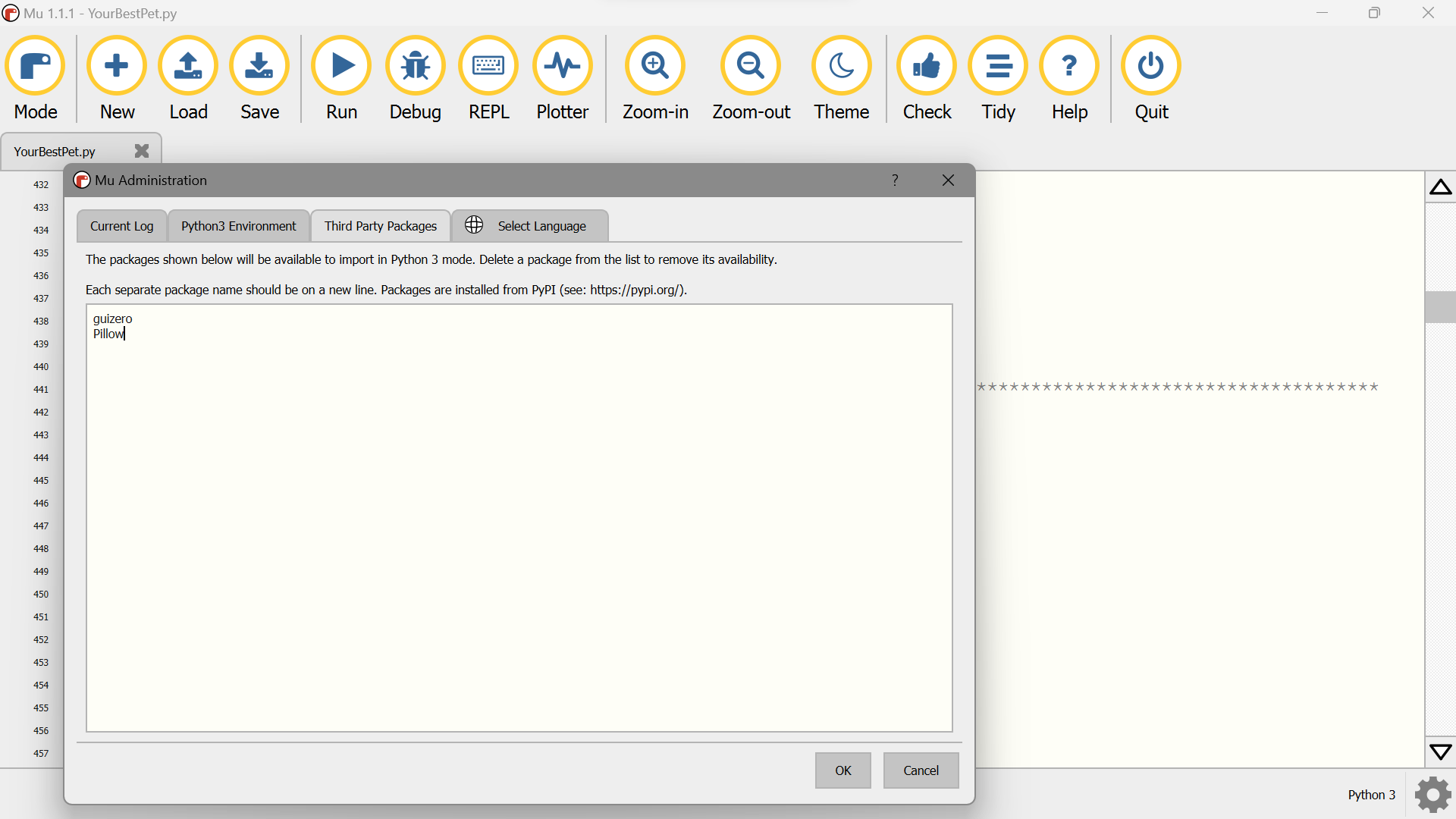
However, since the installation and configuration of such an advanced IDE may be troublesome for a new user and is not necessary for smooth code reading/operation, below are the steps for creating a functional programming environment based on the free “Mu Editor” program.

Please note, that this steps were tested only on Windows operating system and may not be compatible with any other systems such a Linux or MAC.

* Install Python on the computer if it is not already installed. This can be done from https://www.python.org/downloads/
* Install the required libraries, Pillow and tkinter, using the following command in the command prompt or terminal: pip install Pillow tkinter or pip install Pillow guizero

If the user is on Windows, they can open the command prompt by pressing Windows key + R, typing "cmd", and pressing enter

* Install an integrated development environment (IDE) such as MuEditor. This can be downloaded from https://codewith.mu/en/download
* Open MuEditor and click on "Load" to open the .py file.
* Click on "Run" to execute the program.

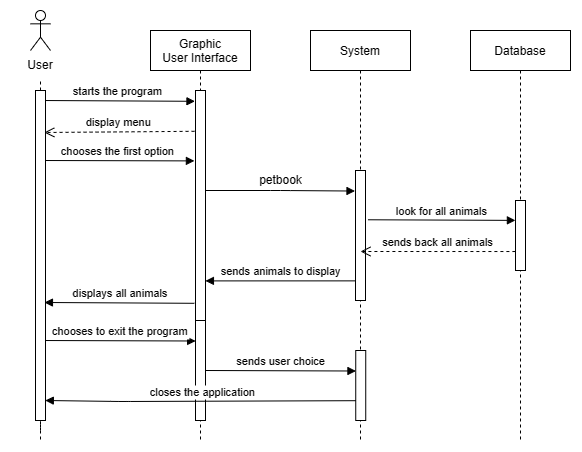


In case of problems running the code, please click on the Settings icon in the lower right corner of the screen. Then go to the “Third Party Packages” tab. There, enter "guizero" and "Pillow" in sequence to download specific packages and click the OK button (each package on a new line). Another window will be displayed, in which you should wait for the additional downloads to complete. When they are installed, the program will display a message. Now you can close the window and run the code again with the “Run” button.

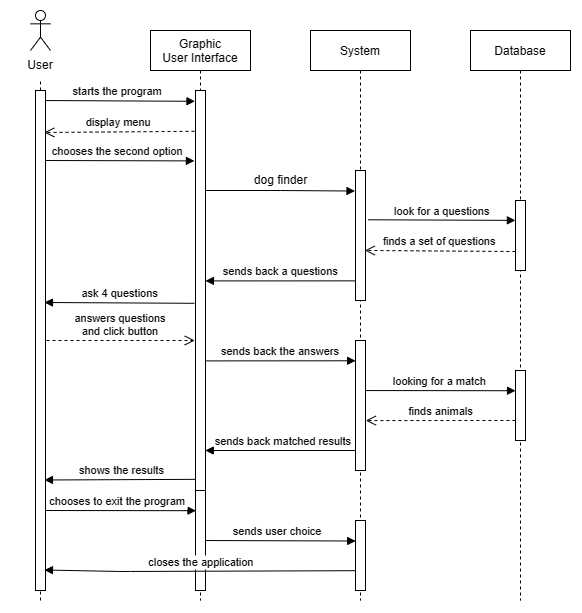
# Updated UML diagrams

## Sequence diagrams

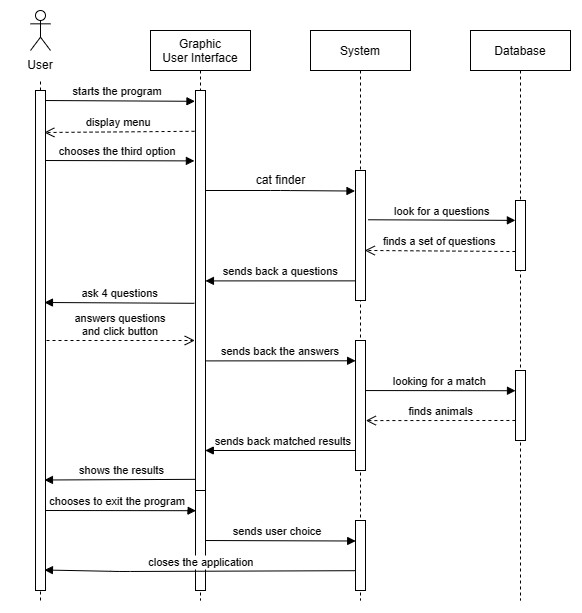
### Petbook



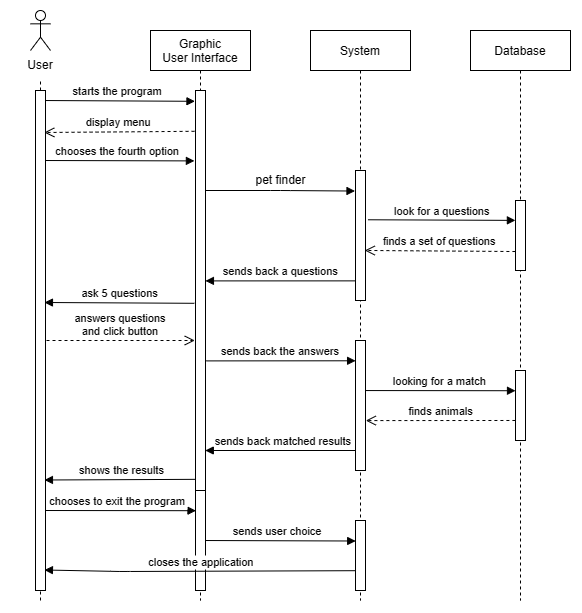
### Dog finder



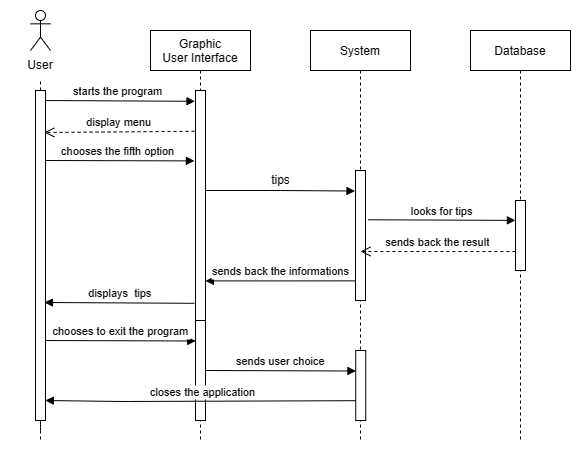
### Cat finder



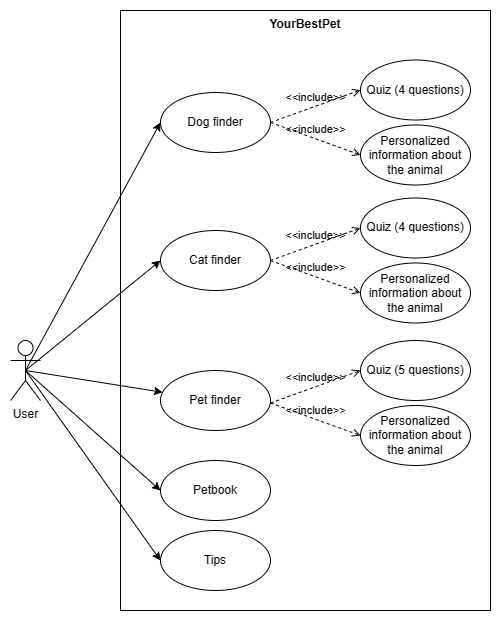
### Pet finder



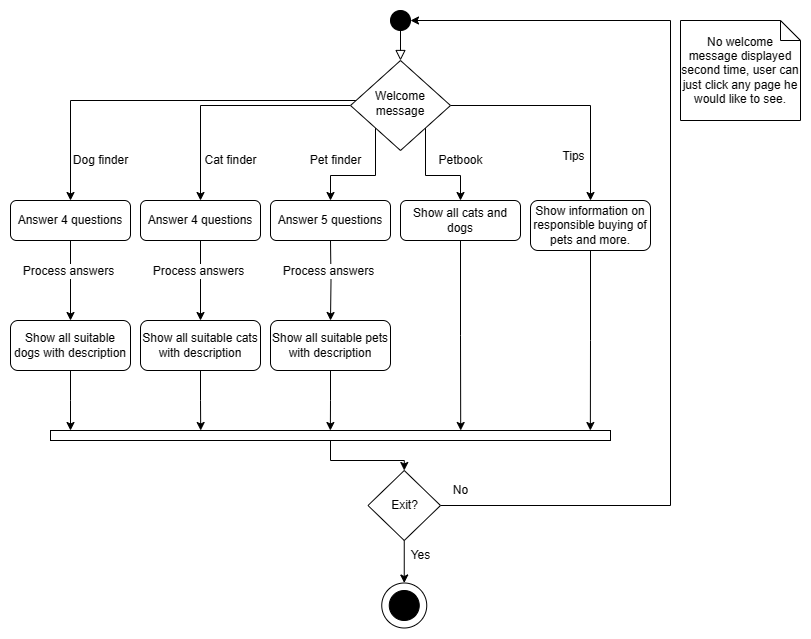
### Tips



## Use case diagram



## Activity diagram



# New Logo

## Main



## Icon



# Client requirements checklist

## Coding of the problem domain

### Does the code match the design?

The issue was discussed with the client before the final version of the product was delivered. It was agreed that any changes made during the development process were a normal part of the creative process and remained consistent with the original project, improving it rather than completely modifying it.

The program has all the initially assumed functions, sometimes modifying the access to them to make it more user-friendly.

### Does the code make appropriate use of inheritance and/or associations?

Inheritance was used to create the Animal class, and then subclasses Dog, Cat, Rabbit, Rat, and Fish. These subclasses have many attributes in common with the mother "Animal" class, such as "size", but also their own attributes such as "general\_description", assigned not to individual objects but to the general description of the entire class, such as the description of a Dog as a breed in general, which is needed later for the "Pet finder" program function. This also enables easy separation of dogs and cats.

### Does the code make appropriate use of scope operators?

In some functions, the "global" option was used to access specific parameters throughout the entire code. In other cases, the functions use local variables that are created inside the function, only to find specific animals in the database (such as the Dog Finder function).

### Does the code make appropriate use of setters and getters and/or properties as appropriate?

The program retrieves different information from classes in various parts of the code in order to modify and/or display them (such as displaying the name of the "Fish" class when it is selected based on the algorithm or modifying the "score" of the Dog class to find the best match).

### Does the code make appropriate use of constructors?

The constructor "\_\_init\_\_(self)" was used in the Animal class, along with assigning specific parameters to it. Additionally, the subclasses were correctly linked to the parent class. Use of "super().\_\_init\_\_()" is not important for this particular code, and it wasn’t used. The constructors have been used appropriately and in the required context, only in the places where they are necessary. The program has been tested to ensure their proper functioning.

### Are the methods and attributes coded correctly

Attributes have been assigned correctly and checked for their functionality. Methods were not used, as the client's requirements did not specify that the entire code should be written solely using object-oriented programming. Therefore, external functions are responsible for modifying and accessing the class attributes.

## Coding of the UI domain

### Does the solution match the design?

The issue was discussed with the client before the final version of the product was delivered. It was agreed that any changes made during the development process were a normal part of the creative process and remained consistent with the original project, improving it rather than completely modifying it.

The program has a slightly different design and graphical interface for better clarity, but it remains consistent with the initial concept. Each function has a separate window as was stated in the initial documentation, and each animal is displayed according to the conceptual graphics (image, title, description).

### Appropriate use of UI controls to ensure that the data entered is valid (eg validation routines and/or error prevention).

The program has been written with the utmost care for clarity and simplicity, excluding as much as possible the potential for errors that could be caused by poorly designed UI (e.g. unnecessary text input windows or the ability to edit displayed data). All possible combinations of buttons have been thoroughly tested. Examples of error handlers are not error messages, but rather methods that prevent them, such as the "Find a dog!" button, which remains gray and inactive until all answers are provided. More informations in “Error handling” part of this document.

### Coding of appropriate events.

The program has many functions that are triggered by interacting with the graphical interface.

### Does the code link appropriately to the business model?

The GUI reflects the business model that was designed in the planning stage. All UI options lead the user to specific functions, which are based on the internal code that searches the internal database and returns specific results (like breed description or pictures), which are then displayed by the GUI. That means the view model and business model work together throughout the entire operation of the program.

## Use of unfamiliar libraries and/or constructs

During the development of the program, several previously unknown libraries, programs, and websites were used. The following is a list of some of them, along with their descriptions and how I used them to create the program:

* tkinter - GUI, extended version compared to GuiZero, allowed for the creation of scrollable windows, not just text fields, which was crucial in this program
* Pillow – opening and resizing images
* PyInstaller – create an executable file
* Canva - logo design
* InstallForge - creation of an installation wizard
* Bing Image Creator - creation of pictures for all animals
* GitHub Copilot - IDE integration, enabling smooth writing of repetitive parts of code

Some of the aforementioned tools, such as tkinter and Pillow, were just additional libraries, but understanding how to use them throughout the entire program required a lot of research and technical documentation. PyInstaller, on the other hand, is a library that consists almost entirely of unknown and inclusive commands used to create an executable file. Canva and InstallForge were programs I had never used before. Canva has many functions that required me to understand the basics of the program. InstallForge, on the other hand, was a more complicated tool, as I had to set up every part of the installer myself, such as the application icon, destination folder for the files, which files should have a desktop shortcut, user options during installation, other graphics, etc. Bing Image Creator required me to find the appropriate key for entering prompts, and GitHub Coopilot required me to register with the service and understand the instructions for how to integrate it with my IDE - PyCharm.

In the "YourBestPet - developer version" folder, there is a file named "PY to Exe converter.py". This is a program that I created for the purpose of this project to generate a file with the .exe extension, which is necessary for the later part (creating an installer). I am leaving it in the folder as evidence of using the PyInstaller module. However, I want to point out that the version that is in the folder is not universal, as indicated by the GUI (you would need to remove the line that assigns an icon, or possibly also the "no console" line if the future program doesn't have its own separate GUI). Additionally, attempting to generate YourBestPet.exe again using this code will create a new .exe version of the application, but if the user has not properly imported all the libraries and set up the interpreter, the generated file may display an error upon running, as not all libraries will be compiled into it, meaning it won't be able to run and understand itself.

Obraz zawierający tekst

Opis wygenerowany automatycznie Obraz zawierający tekst

Opis wygenerowany automatycznie Obraz zawierający tekst, zrzut ekranu, monitor, srebro

Opis wygenerowany automatycznie

There are evidences of using the tkinter, Pillow and PyInstaller libraries in the code because they were used there. Evidence of using the Bing Image Generator can be seen in the "Dogs" and "Cats" folders, where the generated images of original size are located - there is always a Bing creator icon in the lower left corner. Evidence of using "InstallForge" and GitHub Coopilot can be seen in the screenshots above.

## Error handling

The program, as described above, was created to minimize the possibility of errors. One of the example view models is the one provided in an earlier chapter of this document. However, to facilitate finding a few other examples for someone who is not familiar with the code, I will list a few examples that cover both the view and business model of this application.

An important part of the code that eliminated potential errors was separating the user's responses into (for example) answer1\_dog, answer1\_cat, and answer1\_pet. Previously, this was a single list of answers, which would give good results, but would reset the progress of the previous search and cause an error if, for example, the user did the dog quiz first and then the cat quiz. In other words, all variables related to quizzes have been separated into three distinct categories and are independent of each other.

The answers were also divided based on the questions, so that the user could answer in random order (when the answers were previously a single list, the program did not know which answer was given for which question, so it could not provide tailored advice based on them).

Additionally, to avoid another issue with different tabs of the program, additional variables have been introduced, such as welcome\_message\_opened, which tracks which window is currently open and hides other windows based on it.

The above example has been supplemented with two additional error handlers. One is the fact that all windows are hidden at the start and have a default status of False (indicating that they are in a state of being hidden). The second one is that quiz result cards are not simply hidden every time, but completely destroyed, so that the data can be overwritten with the latest results (in the previous version, the results overlapped, causing errors). Quiz tabs (not answers) are only hidden, so that the user can see the last answer - which is indicated by a modified title when they enter a quiz more than once.

Another typical error handler is to protect against situations where the user's answers (in a quiz) do not match any animal in the database. Such a case can be observed by answering the each first answer in the Dog Finder quiz. Since the user indicates that they have children, the list of matches is empty. In this case, after clicking the "Find a dog!" button, it changes its name to "No results, try again!", and the window with answers is not displayed. To achieve this, the program checks whether the list of results is not empty.

Another example is creating two separate lists for Dog and Cat Finders. These are the lists of best\_dog (or cat)\_fits and new\_best\_dog (or cat)\_fits. This is important because going through the list while removing items can cause errors (such as removing dogs/cats that are not suitable for children, if needed).

The console is also an important element of the developer version of the program, where results from each quiz are displayed in an extended form in real-time. After answering the questions and choosing to go to the results, every time in the console, we can see our answers, best fits, and all other animals along with their scores, to ensure that the algorithms are working correctly. This applies to all quizzes in the program.

The last example I will mention is not displaying the "Why this pet is not good for you" list if it is empty.

There are more examples, but the ones described above should demonstrate the attention to detail with which the code was written.

## Internal documentation

The code is commented throughout its entire length in a clear manner and with certain assumptions that make the whole cohesive, and specific elements of the code clearly separated and easy to find. Additionally, it has been divided into sections such as:

* the preamble (title, author, description)
* imports
* classes
* lists
* global variables
* functions (also divided into subsections depending on their function)
* GUI (divided into title bar, then all the specific program tabs, and finally the footer)
* main loop

The sections have been separated in the code using a line consisting of the "\*" character.

The subsections, that is, all the separate program tabs, have also been separated using a line consisting of the "-" character.

Each important line of code introducing something new has been described with an appropriate comment, and any complex functions additionally have their longer description at the beginning of each of them.

All variables and functions that have similar purposes but have been separated for certain technical reasons (such as separate quiz functions for cats and dogs) have corresponding, similar names for their common elements, with the additional word "cat" or "dog" added to the specific element. This is intended to quickly find the general function/variable and then narrow down the results. An example of such functions are: show\_pet\_finder(), show\_cat\_finder(), and show\_dog\_finder(). An example of such variables are: button\_dog\_1a, button\_cat\_1a, and button\_pet\_1a.

The program uses the same colours and font types/sizes throughout to maintain maximum consistency, both in the business and the view model.

For more information about the functions used in the application, please read testing part of this document and internal documentation (code itself).

# Testing strategies

## Test objectives

The application has been tested to meet functional and non-functional requirements, which were initially formulated in the documentation created during the planning stage. Taking these into account, as well as using the MoSCoW method from the planning stage, and modifying them based on changes made during the development stage (adding or modifying some functions of the program to improve it overall), a new list of assumptions was created that the product must meet.

Here are the assumptions of the final version of the application that were tested to ensure that all of them were met:

### Design

* The application is divided into three sections - a title, a working section, and a footer.
* App has a well-designed logo
* Well-designed, clear and consistent interface

### Data

* Separate sets of questions, depending on whether we are looking for a species or breed
* A personalized set of tips and information based on user responses
* Description of 5 different species of animals
* Description of 15 species of cats
* Description of 15 species of dogs
* Pictures of all cats and dogs

### Functionality

* Welcome page
* 5 additional pages: Petbook, Dog finder, Cat finder, Pet finder, Tips
  + 3 pages with quizzes
    - These pages lead to a set of interactive quizzes
    - Error handler that checks if all questions answered
    - Answering the questions generates the choice of species / breed
    - The program displays information about the species / breed selected for the user
    - The program displays specific tips based on answers for each question
    - Returning to any of quizzes shows last answers and different title
  + Pet encyclopaedia with all cats/dogs
    - Option displays a database of dogs and cats
    - Cats and dogs separated
  + Tips for pet owners
    - Option displays additional information and tips

### Non-functional

* Professional - a program created according with ethic standards and all software licensing acts
* Usability - the program will be useful to people interested in buying a new pet and will broaden knowledge and good practices in animal care
* Time management - completion of the project in the required time
* Stability - maintaining code transparency, selection of appropriate functions and tools
* Performance – it is a small program that runs well even on old devices
* Security - the program does not store sensitive user data and does not cause any errors, does not interfere with computer components in an undesirable way
* User friendly - maintaining consistency, app is well designed
* Cost – program is not expensive to make
* Target - achieving all goals stated in the documentation for Graded Unit 2
* Free - creating a program available to everyone for free

## Test environment

For testing purposes, two separate computers were used. One was the primary computer on which the program was developed, and the other was an additional computer simulating an end user. The additional computer did not have any programming environment pre-installed, allowing the program to be tested in both consumer and developer versions for functionality and clarity of instructions. In addition, both computers have different technical specifications, with one having significantly lower power, allowing the performance and stability of the application to be tested. Both units run on the Windows operating system, have a full HD resolution screen, and have constant (though not required) internet access. Third-party libraries used to run developer version are tkinter and Pillow.

## Test strategy

As the program needs to be thoroughly tested from both the business and view model perspectives, many different types of testing strategies have been implemented, which are listed below.

I began the testing process with white box testing to test the internal logic of the software and ensure that the code is working as expected. This helped me catch any errors or bugs early in the development process, making them easier to fix. I checked the overall code readability, its internal documentation and structure, and scanned the entire code using PyCharm to look for any inconsistencies or typos.

Next, I used unit testing to test individual units of the software in isolation to ensure that they are functioning correctly. This allowed me to catch any errors or bugs that might not be apparent from white box testing alone. All functions have been tested separately and then combined together in one whole system.

After that, I used black box testing to verify the software's functionality from an end-user perspective without any knowledge of the internal workings of the program. This helped me identify any issues or errors that could impact the user experience. One person helped me with this test to make sure that all errors would be noticed. This way, I gained an additional perspective and doubled my chances of finding any potential errors. It also showed me whether my program is user-friendly for the end-user who doesn't know coding and only uses the application without any instructions.

Performance testing was then used to test the software's performance under various conditions, such different computers and apps opened in the backgrounds. This helped me ensure that the software can handle the expected workload and that it responds quickly and efficiently.

Usability testing was used to test the software's ease of use and user interface design. This helped me ensure that the software is intuitive and easy to use, meeting the necessary requirements for usability.

Conformance testing was used to ensure that the software meets specific industry standards or regulatory requirements, like using an animal pictures only if I can do it, same as all other data.

I plan to deliver the software for user acceptance testing by the client to ensure that the software meets their specific requirements and is suitable for their needs. This will allow me to make any necessary changes or improvements for the future versions of the product as well as gain a mark needed to pass Graded Unit Exam.

It can be seen that I have performed many tests on both the business model (white box, unit testing) and the view model (black box, usability). In addition, I have added tests aimed at verifying whether the product in this state meets the customer's requirements as well as legal requirements.

The above description pertains to the final tests. However, it should be remembered that when writing the code, I had in mind many of the above requirements, so that the product I created was consistent with the intention and did not require major changes. During the product development, I showed its individual parts (e.g. separate pages) to a person close to me for evaluation of its appearance and functionality from the perspective of black box testing and usability. When it did not meet the requirements, I improved its design. If a part of the code turned out to be not working correctly, I conducted unit testing of the responsible functions and repaired the underlying business model to meet all requirements (both visual and functional related).

The final chapter of the document contains the tests conducted by me to verify the quality of the product. It is evidence of conducting black box, white box, and unit testing. I also made sure that user acceptance testing will be passed by conducting it myself based on the test objectives (listed before). I conducted performance and usability testing as well, checking them on my own and giving them for evaluation to the aforementioned person.

# Test Plan/Test Log

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Item Tested | Data Used | Expected Result | Actual Result | Action Taken | Screenshot |
| View model | | | | | |
| In this section, all visual aspects of the final product are tested. This includes the appearance of individual program pages, changes in buttons colours and names, correct display of data from the database, such as descriptions or photos, and the readability and consistency of all elements, such as fonts, margin sizes, and spacing between paragraphs. This also tests all options of external libraries used to create the application, such as tkinter (e.g. button, canvas, label, scrollbar, forget, destroy) and Pillow (e.g. load, resize).  Some of the tests also confirm the proper work of other functions, but these will be tested separately in the business model using unit testing. The same applies to the correctness of the results presented to the user. | | | | | |
| Welcome Message | Program Launch | A welcome message is displayed in the middle section of the application | The welcome message is displayed as expected | Pass |  |
| Petbook | The user selects Petbook from the list of options located in the program footer. | A list of all dogs and cats with their photos, titles, and short descriptions is displayed in the middle section of the application with a scrollbar on the right side to scroll up and down, Petbook button changes colour | The list of dogs and cats is displayed as expected with the scrollbar working properly and different button colour | Pass |  |
| Dog Finder | The user selects Dog finder from the list of options located in the program footer | The list of questions along with answer buttons is displayed in the working area of the screen. The "Find a dog!" button is inactive. The colour of the "Dog finder" option located in the program footer changes, while all other options have a lighter shade. | The list of questions along with answer buttons is displayed in the working area of the screen. The "Find a dog!" button is inactive. The colour of the "Dog finder" option located in the program footer changes, while all other options have a lighter shade. | Pass |  |
| Dog Finder (3 answers) | User answers three questions with provided options, changes one of the answers, but don’t answer for the last question | The answer buttons change colours as the answers are given. Changing an answer causes the leading button colour in that question to change. The "Find a dog!" button remains grey and clicking it does nothing. | The answer buttons change colours as the answers are given. Changing an answer causes the leading button colour in that question to change. The "Find a dog!" button remains grey and clicking it does nothing. | Pass |  |
| Dog Finder (4 answers) | User answers four questions with provided options | The button below the questions changes colour and leads to a page displaying the best-suited dogs (become active). This happens only after answering all the questions. | The button changes colour as expected and become active. | Pass |  |
| Dog Finder (no match) | The user answers the four questions in a way that no match is found. Then click a button “Find a dog!”. | The "Find a dog!" button changes colour (since all the answers have been given), but when clicked, its name changes to "No results, try again!". Clicking the button without changing the answers doesn't have any additional effects. | The "Find a dog!" button changes colour (since all the answers have been given), but when clicked, its name changes to "No results, try again!". Clicking the button without changing the answers doesn't have any additional effects. | Pass |  |
| Dog Finder (results) | User answers four questions with provided options.  The answers have been selected to ensure that matches will be found. Then click a button “Find a dog!”. | The button below the questions changes colour and leads to a page displaying the best-suited dog for the user based on their answers with a photo, title, and short description along with the user's answers and explanation for why that dog fits or doesn't fit the user's preferences | The button changes color as expected, and the dog's page is displayed with the user's answers and explanation | Pass |  |
| Dog Finder (clicked again after previous results) | Dog Finder clicked again after previous results. | The "Dog finder" card now has a different title. It is "Wanna try again? Your last answers:" now. In addition, the previous answers are pre-selected by default, and the "Find a dog!" button is active (since all the answers have been given). The user can change the answers before clicking the button again. | The "Dog finder" card now has a different title. It is "Wanna try again? Your last answers:" now. In addition, the previous answers are pre-selected by default, and the "Find a dog!" button is active (since all the answers have been given). The user can change the answers before clicking the button again. | Pass |  |
| Cat Finder | The user selects Cat finder from the list of options located in the program footer | The list of questions along with answer buttons is displayed in the working area of the screen. The "Find a cat!" button is inactive. The colour of the "Cat finder" option located in the program footer changes, while all other options have a lighter shade. | The list of questions along with answer buttons is displayed in the working area of the screen. The "Find a cat!" button is inactive. The colour of the "Cat finder" option located in the program footer changes, while all other options have a lighter shade. | Pass |  |
| Cat Finder (3 answers) | User answers three questions with provided options, changes one of the answers, but don’t answer for the last question | The answer buttons change colours as the answers are given. Changing an answer causes the leading button colour in that question to change. The "Find a cat!" button remains grey and clicking it does nothing. | The answer buttons change colours as the answers are given. Changing an answer causes the leading button colour in that question to change. The "Find a cat!" button remains grey and clicking it does nothing. | Pass |  |
| Cat Finder (4 answers) | User answers four questions with provided options | The button below the questions changes colour and leads to a page displaying the best-suited cats (become active). This happens only after answering all the questions. | The button changes colour as expected and become active. | Pass |  |
| Cat Finder (no match) | The user answers the four questions in a way that no match is found. Then click a button “Find a cat!”. | The "Find a cat!" button changes colour (since all the answers have been given), but when clicked, its name changes to "No results, try again!". Clicking the button without changing the answers doesn't have any additional effects. | The "Find a cat!" button changes colour (since all the answers have been given), but when clicked, its name changes to "No results, try again!". Clicking the button without changing the answers doesn't have any additional effects. | Pass |  |
| Cat Finder (results) | User answers four questions with provided options.  The answers have been selected to ensure that matches will be found. Then click a button “Find a cat!”. | The button below the questions changes colour and leads to a page displaying the best-suited cat for the user based on their answers with a photo, title, and short description along with the user's answers and explanation for why that cat fits or doesn't fit the user's preferences | The button changes colour as expected, and the cat's page is displayed with the user's answers and explanation | Pass |  |
| Cat Finder (clicked again after previous results) | Cat Finder clicked again after previous results. | The "Cat finder" card now has a different title. It is "Wanna try again? Your last answers:" now. In addition, the previous answers are pre-selected by default, and the "Find a cat!" button is active (since all the answers have been given). The user can change the answers before clicking the button again. | The "Cat finder" card now has a different title. It is "Wanna try again? Your last answers:" now. In addition, the previous answers are pre-selected by default, and the "Find a cat!" button is active (since all the answers have been given). The user can change the answers before clicking the button again. | Pass |  |
| Pet Finder | The user selects "Pet finder" from the menu. | The list of questions and corresponding answer buttons are displayed. The "Pet finder" button in the menu is highlighted with a different colour. Also, the "Find a pet!" button is disabled. | The list of questions and corresponding answer buttons are displayed. The "Pet finder" button in the menu is highlighted with a different colour. Also, the "Find a pet!" button is disabled. | Pass |  |
| Pet Finder (4 answers) | User answers four questions with provided options, changes one of the answers, but don’t answer for the last question | The answer buttons change colours as the answers are given. Changing an answer causes the leading button colour in that question to change. The "Find a pet!" button remains grey and clicking it does nothing. | The answer buttons change colours as the answers are given. Changing an answer causes the leading button colour in that question to change. The "Find a pet!" button remains grey and clicking it does nothing. | Pass |  |
| Pet Finder (5 answers) | User answers five questions with provided options | The button below the questions changes colour and leads to a page displaying the best-suited pets (become active). This happens only after answering all the questions. | The button changes colour as expected and become active. | Pass |  |
| Pet Finder (results) | User answers five questions with provided options.  Then click a button “Find a cat!”. | The button below the questions changes colour and leads to a page displaying the best-suited pet for the user based on their answers. | The button changes colour as expected, and the result page is displayed with the description. | Pass |  |
| Pet Finder (clicked again after previous results) | Pet Finder clicked again after previous results | The "Pat finder" card now has a different title. It is "Wanna try again? Your last answers:" now. In addition, the previous answers are pre-selected by default, and the "Find a cat!" button is active (since all the answers have been given). The user can change the answers before clicking the button again. | The "Pat finder" card now has a different title. It is "Wanna try again? Your last answers:" now. In addition, the previous answers are pre-selected by default, and the "Find a cat!" button is active (since all the answers have been given). The user can change the answers before clicking the button again. | Pass |  |
| Tips | None | A page with general tips for pet owners is displayed in the middle section of the application, scrollbar works, “Tips” have a darker shade than all other menu options. | The tips page is displayed as expected, including working scrollbar and button of different colour. | Pass |  |
| Resizing the app window. | Attempting to resize the window, then minimizing and restoring the application, moving the window and finally using the "x" button to close the application. | The application does not allow resizing the window, but it does allow minimizing it. Restoring it from the minimized state in the taskbar does not affect the performance of the application or change its elements. The same applies to moving the window around. Clicking the "x" button closes the window and the entire application. | The application does not allow resizing the window, but it does allow minimizing it. Restoring it from the minimized state in the taskbar does not affect the performance of the application or change its elements. The same applies to moving the window around. Clicking the "x" button closes the window and the entire application. | Pass | N/A |
| Business model/Functions | | | | | |
| In this section, all functions of the program and the correctness of output data will be tested. | | | | | |
| hide\_welcome\_message() | User click on any button in the footer (menu) | Welcome message disappear | Welcome message disappear | Pass |  |
| show\_pet\_finder() | These functions are used to reduce the length of repetitive code. As they always function as subfunctions of the open\_...() functions, it is not necessary to test them separately, if all open\_...() functions - tested next - work correctly. The show\_...() and hide\_...() functions are responsible solely for showing/hiding their assigned card (e.g. "dog finder") and changing the colour of its button. The open\_...() functions use these functions to hide/show individual cards, displaying only the selected one. | | | | |
| hide\_pet\_finder() |
| show\_pet\_finder\_answers() |
| hide\_pet\_finder\_answers() |
| show\_dog\_finder() |
| hide\_dog\_finder() |
| show\_dog\_finder\_answers() |
| hide\_dog\_finder\_answers() |
| show\_cat\_finder() |
| hide\_cat\_finder() |
| show\_cat\_finder\_answers() |
| hide\_cat\_finder\_answers() |
| show\_tips() |
| hide\_tips() |
| show\_petbook() |
| hide\_petbook() |
| open\_dog\_finder() | The user opens the dog finder from the bottom menu of the application. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | Pass |  |
| open\_cat\_finder() | The user opens the cat finder from the bottom menu of the application. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | Pass |  |
| open\_pet\_finder() | The user opens the pet finder from the bottom menu of the application. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | Pass |  |
| open\_tips() | The user opens the tips from the bottom menu of the application. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | Pass |  |
| open\_petbook() | The user opens the petbook from the bottom menu of the application. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | The selected tab is opened and its assigned button changes color. The other buttons and tabs are hidden. | Pass |  |
| The answer...() functions that will be tested below are responsible for changing the colour of buttons and saving answers. Each button in each question and quiz is assigned a separate function.  This is done to allow:   * saving previous answers for each quiz separately (for later reminder to the user) * answering questions in any order (each question has a separate variable, such as answer1\_dog, answer2\_dog, etc. and each function overwrites the answer only for the specific question) * setting different numbers of possible answers depending on the quiz.   The way these functions are written and the additional information displayed in the console allows them to be tested in pairs without sacrificing results for each one separately. To avoid unnecessarily complicating the documentation, each function is tested by combining them in pairs. It is important that the result of each function (displayed in the console) matches the answer to a specific question, and that the answers displayed in the console correspond to the colour of the buttons. Additionally, after each program page (cats, dogs, pets), the button\_find...() function is tested last, which changes the colour of the bottom button and assigns it a command only if all the answers have been given (shown in another example in view model testing). | | | | | |
| answer\_first\_dog\_question\_A(),  answer\_second\_dog\_question\_A(),  answer\_third\_dog\_question\_A(),  answer\_fourth\_dog\_question\_A() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| answer\_first\_dog\_question\_B(),  answer\_second\_dog\_question\_B(),  answer\_third\_dog\_question\_B(),  answer\_fourth\_dog\_question\_A() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| answer\_first\_dog\_question\_C(),  answer\_second\_dog\_question\_C(),  answer\_third\_dog\_question\_C(),  answer\_fourth\_dog\_question\_B() | All questions answered. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| button\_find\_a\_dog\_ready() | All questions answered. | The button becomes active (assigned a function) and changes colour after answering all the questions. Until then, it remains inactive. | The button becomes active (assigned a function) and changes colour after answering all the questions. Until then, it remains inactive. | Pass |  |
| answer\_first\_cat\_question\_A(),  answer\_second\_cat\_question\_A(),  answer\_third\_dog\_question\_A(),  answer\_fourth\_cat\_question\_A() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| answer\_first\_cat\_question\_B(),  answer\_second\_cat\_question\_B(),  answer\_third\_cat\_question\_B(),  answer\_fourth\_cat\_question\_A() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| answer\_first\_cat\_question\_C(),  answer\_second\_cat\_question\_C(),  answer\_third\_cat\_question\_C(),  answer\_fourth\_cat\_question\_B() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| button\_find\_a\_cat\_ready() | All questions answered. | The button becomes active (assigned a function) and changes colour after answering all the questions. Until then, it remains inactive. | The button becomes active (assigned a function) and changes colour after answering all the questions. Until then, it remains inactive. | Pass |  |
| answer\_first\_pet\_question\_A(),  answer\_second\_pet\_question\_A(),  answer\_third\_pet\_question\_A(),  answer\_fourth\_pet\_question\_A(),  answer\_fifth\_pet\_question\_A() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| answer\_first\_pet\_question\_B(),  answer\_second\_pet\_question\_A(),  answer\_third\_pet\_question\_B(),  answer\_fourth\_pet\_question\_A(),  answer\_fifth\_pet\_question\_A() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| answer\_first\_pet\_question\_C(),  answer\_second\_pet\_question\_A(),  answer\_third\_pet\_question\_C(),  answer\_fourth\_pet\_question\_A(),  answer\_fifth\_pet\_question\_A() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| answer\_first\_pet\_question\_D(),  answer\_second\_pet\_question\_B(),  answer\_third\_pet\_question\_D(),  answer\_fourth\_pet\_question\_A(),  answer\_fifth\_pet\_question\_B() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| answer\_first\_pet\_question\_E(),  answer\_second\_pet\_question\_C(),  answer\_third\_pet\_question\_E(),  answer\_fourth\_pet\_question\_B(),  answer\_fifth\_pet\_question\_C() | The user selects answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | The user's choices are handled by the assigned functions (each button is linked to a specific function). This results in a change of color of the leading buttons and the proper storage of the selected answers. | Pass |  |
| button\_find\_a\_pet\_ready() | All questions answered. | The button becomes active (assigned a function) and changes colour after answering all the questions. Until then, it remains inactive. | The button becomes active (assigned a function) and changes colour after answering all the questions. Until then, it remains inactive. | Pass |  |
| The functions tested below are responsible for gathering the answers saved by previously tested functions, and then assigning scores to each of the animals based on these answers (variables self.fit\_reasons, self.not\_fit\_reasons, self.score for dogs and cats). After assigning the scores, the animals with the highest scores are selected (previously removing those that are not good for children, if applicable) and displayed in a newly opened window. If no animals are matched, the function changes the name of the bottom button to "No results, try again!". To facilitate checking individual elements of the function, the console shows additional data (developer version), allowing you to compare the results of all animals independently and check if the best matches were chosen correctly. Therefore, to consider the functions below as working correctly, the console data must match those displayed by the application, as well as the general attributes of each animal (correctly assigned points). Additionally, everything must be displayed in a clear manner. The visual part has already been evaluated in the view model testing. Nevertheless, all necessary screenshots for evaluation have been provided below. | | | | | |
| find\_a\_dog()  (option 1) | The user chooses a quiz and then answers the questions (the first answer in each of them), keeping in mind that he will not get a match this way. After answering the questions, the user presses the button at the bottom of the screen. | The button at the bottom changes its name to "No results, try again". The console displays the correctly selected answers and scores for all animals in the category. However, due to the lack of matching, the "Best fits" list in the console remains empty. This happens because the only animals that fit the category (small and inexpensive to maintain) are not suitable for children. | The button at the bottom changes its name to "No results, try again". The console displays the correctly selected answers and scores for all animals in the category. However, due to the lack of matching, the "Best fits" list in the console remains empty. This happens because the only animals that fit the category (small and inexpensive to maintain) are not suitable for children. | Pass |  |
| find\_a\_dog()  (option 2) | The user chooses a quiz and then answers the questions. After answering the questions, the user presses the button at the bottom of the screen. | After selecting an answer, the user presses the button at the bottom of the screen. Upon pressing the button, the results are displayed. The interface meets visual expectations, and the results correspond to those predicted (Beagle and Miniature Schnauzer receive maximum scores). The results displayed in the program are confirmed in the console. All of the user's answers are also correctly marked in the console. | After selecting an answer, the user presses the button at the bottom of the screen. Upon pressing the button, the results are displayed. The interface meets visual expectations, and the results correspond to those predicted (Beagle and Miniature Schnauzer receive maximum scores). The results displayed in the program are confirmed in the console. All of the user's answers are also correctly marked in the console. | Pass |  |
| find\_a\_cat()  (option 1) | The user chooses a quiz and then answers the questions (the first answer in each of them), keeping in mind that he will not get a match this way. | The button at the bottom changes its name to "No results, try again". The console displays the correctly selected answers and scores for all animals in the category. However, due to the lack of matching, the "Best fits" list in the console remains empty. This happens because the only animals that fit the category (small and inexpensive to maintain) are not suitable for children. | The button at the bottom changes its name to "No results, try again". The console displays the correctly selected answers and scores for all animals in the category. However, due to the lack of matching, the "Best fits" list in the console remains empty. This happens because the only animals that fit the category (small and inexpensive to maintain) are not suitable for children. | Pass |  |
| find\_a\_cat()  (option 2) | The user chooses a quiz and then answers the questions. After answering the questions, the user presses the button at the bottom of the screen. | After selecting an answer, the user presses the button at the bottom of the screen. Upon pressing the button, the results are displayed. The interface meets visual expectations, and the results correspond to those predicted (Sphynx and Scottish fold receive maximum scores). The results displayed in the program are confirmed in the console. All of the user's answers are also correctly marked in the console. | After selecting an answer, the user presses the button at the bottom of the screen. Upon pressing the button, the results are displayed. The interface meets visual expectations, and the results correspond to those predicted (Sphynx and Scottish fold receive maximum scores). The results displayed in the program are confirmed in the console. All of the user's answers are also correctly marked in the console. | Pass |  |
| find\_a\_pet()  (option 1) | The user chooses a quiz and then answers the questions. After answering the questions, the user presses the button at the bottom of the screen. | After answering the questions and clicking the button, the result screen is displayed as expected (dog). The answers are correctly marked in the console and the score is calculated accurately. | After answering the questions and clicking the button, the result screen is displayed as expected (dog). The answers are correctly marked in the console and the score is calculated accurately. | Pass |  |
| find\_a\_pet()  (option 2) | The user chooses a quiz and then answers the questions. After answering the questions, the user presses the button at the bottom of the screen. | After answering the questions and clicking the button, the result screen is displayed as expected (rat, rabbit). The answers are correctly marked in the console and the score is calculated accurately. | After answering the questions and clicking the button, the result screen is displayed as expected (rat, rabbit). The answers are correctly marked in the console and the score is calculated accurately. | Pass |  |