A JOURNEY TO BUILD A DOG WALKING WEB APPLICATION

by

ANGELA PELKY

A THESIS

Presented to the Department of Computer Science
and the Robert D. Clark Honors College
in partial fulfillment of the requirements for the degree of
Bachelor of Science

May 2023

An Abstract of the Thesis of

Angela Pelky for the degree of Bachelor of Science
in the Department of Computer Science to be taken June 2023

Title: A Journey to Build a Dog Walking Web Application

Approved: ------ *Title and Full Name (or Title at end of Full Name [i.e., Ph.D.])*
Primary Thesis Advisor

For those who are new to the world of computer science, what are your thoughts? Is it intimidating? Is it a black box? Is it something that will eventually have robots taking over the world? Well, I am here to tell you that it is all of those things and none of those things if you look at it through the right lens. Computer science background or not, this paper is meant for any person who would like to build their own, independent business platform. The use case example of building my own dog walking web app will exemplify two key facts. First, it will provide a code template for any small business owner to use and make their own. Second, it will exemplify the asset and tool that computer science can be without having a formal education or background in the subject.

Acknowledgements

It has been such an honor and a privilege to work with a number of people on this project. I would like to thank my thesis advisors, Eric Wills and Lindsay Hinkle for providing me with the guidance and confidence I needed to develop this project. I would like to thank my family for shaping me into the person I am today. Finally, I would like to thank my partner and friends for supporting me and inspiring me. I feel so lucky to have such amazing people in my life.

Table of Contents

List of Figures

System Requirements Specification

1. The Concept of Operations (ConOps)

The main objective of this system is to create a web application known as Angela Walks Dogs designed for dog owners who are in need of dog walking or pet sitting services. Angela Walks Dogs provides functionality for dog owners to book services with caregiver, Angela. Furthermore, it allows users to pay for services and to view upcoming services on a single interface.

1.1. Current System or Situation

According to Spots.com, as of April 2023, 63.4 million American households own a dog [1]. Based on an estimate that 37.8% of Oregonians own a dog, that would amount to 65,000 potential customers in Eugene alone [2]. So, the justification for building a web app that supports a dog walking business is strong.

This is not an untapped market, though. Common competitors in the space are Wag! Labs Incorporated and Rover.com. According to WagWalking.com, Wag! Labs Incorporated is an app that provides a multitude of services [3]. These services include: pet training, pet sitting, pet boarding, walks, and drop-in visits. The pet parents can either book a service in advance or request an on-demand service that will be fulfilled within the next hour. Something unique about Wag! is its process of requesting and booking. When a pet parent decides they need a service completed, they will create the specification of the service and then every pet caregiver in the area will be notified of this potential service. Once notified, a pet caregiver will “request” the services. From there, the pet parent can decide which pet caregiver they would like to come watch or walk their pet. The Wag! app also provides interesting functionality such as tracking the walker during the service.

Another competitor is Rover, which can be accessed through both mobile app and desktop. According to Rover.com, they offer a variety of services such as: pet boarding, house sitting, dog walking, day care, drop-in visits, and dog training [4]. Unlike Wag!, when a pet parent is looking for a service they can scroll through pet caregivers in the area. Once they have found someone they like, they will send them a message about their pet and the service they would like the caregiver to complete. From there, the caregiver decides if they want to complete the service or not.

Both of these businesses earn their revenue by taking a commission from each service. This is justified because they are providing a network by which pet parents and pet caregivers can connect.

1.2. Justification for a New System

The proposed system, Angela Walks Dogs, hopes to provide a platform that allows dog walkers or any small business owner interested in curating a business, the technological savvy to cut out the middle man. The result would be a web application that allows the business owner (1) a platform to interact and exchange information with customers, (2) ability to connect their business's Google Calendar with the site and offer seamless scheduling services for free, (3) a confirmed bookings page that offers a personalized experience for the customer.

Some limitations of the current systems for pet parents are largely a result of the human element. For example, according to Hepper.com, Wag! underperformed in their availability of on-demand walks and in the pet owners inability to always choose their preferred walker [5]. On the other hand, Caninejournal.com noted that one area where Rover disappointed was in their handling of customer service issues [6].

For caregivers, there are more limitations to consider. As to be expected, the largest con is the cut that Rover and Wag! take. According to NerdWallet.com, Wag! takes 40% of the caregivers earnings and Rover takes 20% of the caregivers earnings [7]. This leads to an inflated expense for the pet parent and a deflated payout for the pet caregiver. Additionally, neither of these businesses allow for personalization. Say, for example, a pet parent prefers text communication versus in-app communication. Neither business allows such modifications.

For pet parents, Angela Walks Dogs will solve many of the problems with a human factor because they will be dealing with the caregiver and the caregiver alone. This takes out any uncertainty or uncomfort because they will know who they are working with and who they are entrusting with their fur baby.

On the other hand, Angela Walks Dogs will provide caregivers freedom and decentralization. The purpose of this system is to be reproducible for any caregiver or small business owner. Therefore, an entrepreneur will no longer have to worry about the tremendous cut that Wag! and Rover take for doing very little work. Not to mention that they will be able to alter the system however they please by simply modifying the code to fit their vision and satisfy their customer segment's needs.

1.3. Operational Features of the Proposed System

With Angela Walks Dogs, there are three main objectives to satisfy the different needs. It will provide functionality for pet parents to: (1) create an account, (2) book services, (3) view their scheduled services.

1.3.1 Web Application

This program is built as a web application. According to GeeksforGeeks.com, a web application is a piece of software that can be accessed by the browser [8]. It uses a combination of server-side scripts and client-side scripts to present information. This is more complex than a website and therefore provides more functionality. An example of a website would be something that presents static information - a restaurant that has a page with its location and phone number and a page with the menu. An example of a web application would be something like Amazon or YouTube.

A web application was chosen for Angela Walks Dogs to increase the functionality of the program. Unlike Wag! which can only be accessed via mobile app, Angela Walks Dogs enhances the user experience by providing flexible access via phone, tablet, or desktop, and no downloads are needed.

1.3.2 Usability

Usability is the most important feature for Angela Walks Dogs because it is directly related to customer satisfaction. According to "How the Website Usability Elements Impact Performance," there are multiple usability elements a website should engineer. These include ease-of-use, information, and interactivity [9].

To ensure ease-of-use, this web application will promote simplicity. With only four pages to navigate through, the user will not have to jump around to find information. One of the downfalls of Wag! and Rover are the numerous pages to navigate through. These pages harbor unnecessary information such as "Wag! Store" or "Refer & Promote" that only clog up space and increase a new user's learning rate. Instead, my web application is engineered to communicate only the most relevant information in a concise and easily understood manner. Furthermore, the application will prioritize interactivity through prompts and personalization; having the user login and view their scheduled services will curate a personalized feel.

Achieving these usability elements are incredibly important for customer retention and relationship management. It has been shown by Aljukhadar and Senecal that "websites with the fastest learning curves receive more visits and realize better business results by achieving higher levels of customer lock-in." So, decreasing the learning curve is crucial for any business platform. Angela Walks Dogs will achieve this by promoting simplicity and communicating only the most relevant information. Furthermore, according to "Customer Loyalty and Customer Relationship Management," customers should be able to readily repeat booking the same service [10]. Angela Walks Dogs enables the customer to easily book the same exact service because there is only one caregiver to choose from and there are only four service options. This system is much more reproducible than Wag! or Rover because both applications require a multi-step booking process. A pet owner needs to (1) find someone they like (2) make sure the caregiver is

available (3) send out the request, compared to Angela Walks Dogs, which only has one step, to book a service. With customer engagement, less is more, so Angela Walks Dogs decreases choice and increases reproducibility.

1.4. User Classes

According to the Institute of Electrical and Electronics Engineers, a user class is defined as a group of people who will interact with the system in a similar manner [11]. For Angela Walks Dogs there are two primary user classes. The first class is pet parents. The pet parents will be interacting with the user interface and will only have access to the user interface. The second user class is small business owners. This user class will manipulate the code in order to satisfy their business needs.

1.5. Modes of Operation

There is only one mode of operation for this system, viewing the web app as a pet parent. But, a pet caregiver or small business owner can download the public GitHub repository and follow this thesis to build their own web app.

1.6. Operational Scenarios/Use Cases

Use Case #1: Create an Account
Process:

1. User opens system via their preferred web browser
2. All pages are loaded
3. User navigates to the “Login” page
4. User scrolls down to the “Register User” section
5. User fills out prompted information
6. User clicks “Register”
7. User will be added to database
8. User receives a “User registered successfully” message

Use Case #2: Schedule a Service
Process:

1. User opens system via their preferred web browser
2. All pages are loaded
3. User navigates to the “Schedule” page
4. User proceeds to “Login” page if not yet authenticated
5. User looks at the Google Calendar and ensures their desired time and date for the service is available
6. System will write to the Google Calendar API with the scheduled service
7. User clicks “Proceed to Payment”

8. System will write new event into database

Use Case #3: View booked services

Process:

1. User opens system via their preferred web browser
2. All pages are loaded
3. User will navigate to the “Booked Page”
4. If user is not yet authenticated, they will navigate to the “Login” page to first login
5. “Booked Page” is populated with user’s stored data
6. User views their upcoming bookings

2. Specific Requirements

This system will need everything under “must need” in order to create a viable prototype that will be (1) reproducible (2) easily built upon.

Must Need:

- A user interface that allows users to navigate through the system
- Access to the internet through a web browser
- Access to real-time calendar information via the Google Calendar API
- A Stripe account for payment processing
- A secure database that will store user information

2.1. External Interfaces (Inputs and Outputs)

2.1.1 User Interface

Purpose: Prompt users with a home page detailing the background and qualifications of their dog walker. From the home page, users will be able to navigate to any other page.

Inputs:

- Login Information: If the user logs in, the system will record the login information
- Scheduling Information: If the user makes a booking while logged in, all specific information will be written to the database

Outputs:

- Booked Information: User will be able to view their booked services if they are logged in

2.2. Functions

2.2.1 User Manager

- The user manager is set up to create a database for storing user information. This will be initiated when a user registers and books a service
- Accepts inputs: username, name, start time, end time

2.2.2 Login

- The login module is built to create a secure database for storing user credentials. It will be initiated when a user tries to (1) login (2) logout (3) register (4) forget username or password
- Accepts inputs: email, username, name, password

2.2.3 Authenticator

- Module that connects to the Google Calendar API to write to and read from the Angela Walks Dogs calendar
- Accepts inputs: Event object

2.2.4 Scheduler

- Allows the user to schedule a service
- Accepts inputs: Service type, username, pet name, address, access instructions, duration of service, date of service, time of service

Software Design Specification

1. System Overview

Angela Walks Dogs is a user interfacing web application that allows pet parents to book services with their walker, Angela. Due to the large user interface component of this web application, it is organized into frontend modules and backend modules.

2. Software Architecture

This software architecture was made to satisfy the requirements of the Python library, Streamlit. Streamlit is structured in such a way that any program you save in a folder called “pages” will automatically create a user interfacing page on your web application. Please see “Introduction to Streamlit” in the appendix for more details.

- Home Page - Simple module that holds information about the owner of the site and the functionality the site provides
- Booked - User interfacing page that displays a user’s scheduled services. Interacts with the login page and the user manager module to extract relevant user information if and only if the user has logged in.
- Login - User interfacing page that allows the user to manage their account. Interacts with a database that stores login information. Calls on helper Streamlit library to perform tasks.
- Scheduler - User interfacing program that allows a user to book a service and pay for a service. Module calls on Authenticator to build a calendar and user manager to store information. Module makes an external call to Stripe for payment acceptance.
- Authenticator - Backend program that loads calendar for Scheduler module
- User Manager - Backend program that builds and manages customer database for Booked, Login, and Scheduler modules.

3. Software Modules

3.1 Home Page - home.py

3.1.1 Role

The role of this page is simple. It is the landing page for the web application, so navigation will default to this page. It also provides a platform where information can be communicated about (1) how to use the web application and (2) who is Angela and why should you let her walk your dog.

3.1.2 Interaction with other modules

This module does not interact with any other modules.

3.2. Booked Page - booked.py

3.2.1 Role

The role of this page is also quite simple. It provides a platform for users to view their upcoming scheduled services.

3.2.2 Interaction with other modules

Booked uses Streamlit's `session_state` functionality to listen to the Login module and wait for the user to login to their account. Once logged in, this module will call on the User Manager module to extract user data.

3.3 Login Page - login.py

3.3.1 Role

The role of this page is more complex than the other two pages. It performs functionality:

1. Logging in
2. Registering account
3. Obtaining a forgotten password
4. Obtaining a forgotten username

Instead of calling on other modules to perform these functionalities, it is isolated to this module. This design decision was made for ease of coding. Once the user inputs their information, we can simply pass along the inputted data into our helper Streamlit library to do the authenticating.

3.3.2 Interaction with other modules

This module interacts with the Booked module and the User Manager module. It sends the "go ahead" to the Booked and Schedule module to inform the program that the user is authenticated and it can display user data. It provides the User Manager with new registered users that can be stored in the customer database.

3.3 Schedule Page - schedule.py

3.3.1 Role

The role of this module is to provide an easy "checkout" page for the user to schedule their services on. It will display a calendar, ask users to fill out a form, and finally navigate to a third party website that will collect user payment.

3.3.2 Interaction with other modules

This module interacts with the Authenticator module to initialize the Google Calendar API. It sends service information to the User Manager module so that it can store user information in the database.

3.4 Authenticator - Authenticator.py

3.4.1 Role

The Authenticator will verify a user's Google credentials and store these credentials within a json file. This functionality is provided to add an extra layer of security to our web application. It will then initialize our Google Calendar API so that we can read and write from the Angela Walks Dogs Calendar.

3.4.2 Interaction with other modules

Information obtained by the Schedule module will be sent to the Authenticator module to write a service to Angela's calendar.

3.5 User Manager - UserManager.py

3.5.1 Role

The User Manager will create a database that stores relevant user information. This information includes username, name, a scheduled event date, start, and end time. It also provides functionality for fetching stored data.

3.5.2 Interaction with other modules

The User Manager interacts with the Login, Schedule, and Booked modules.

4. Models of Operational Scenarios

4.1 User Creates an Account

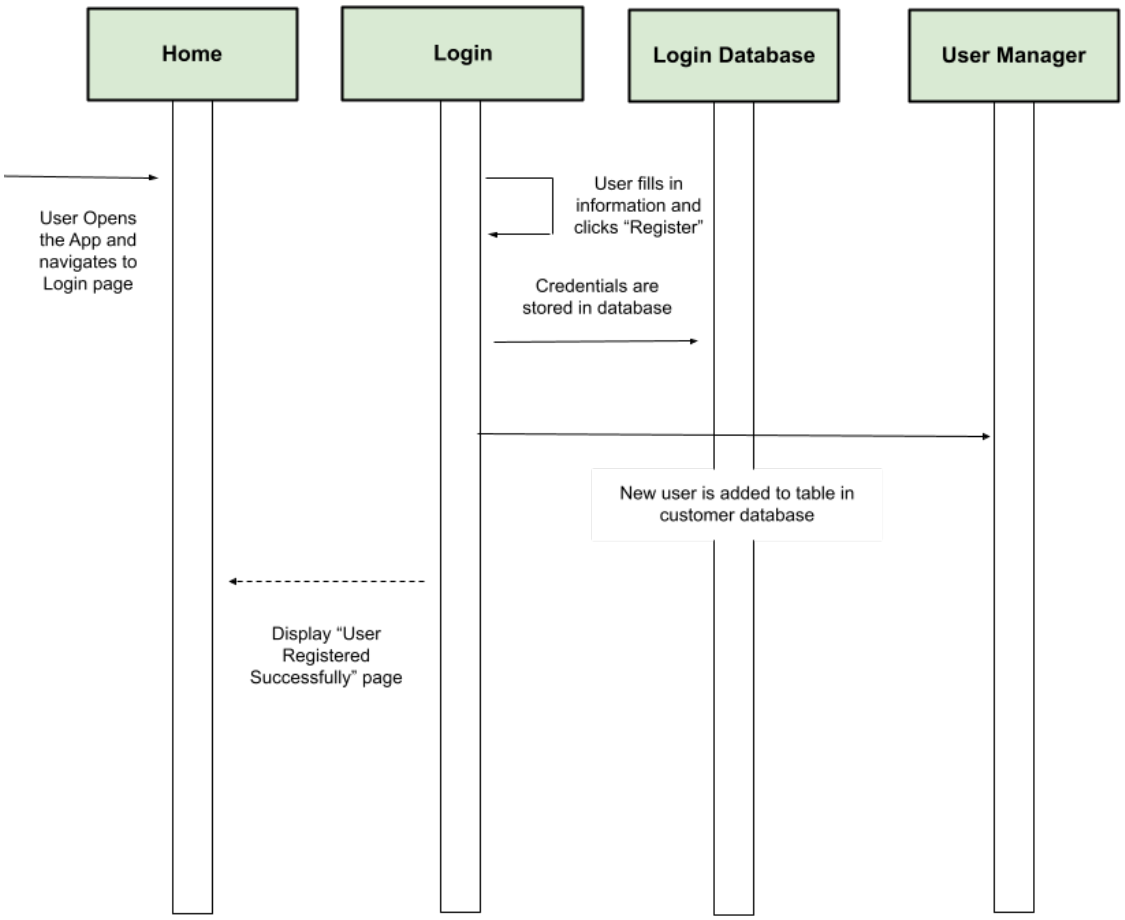

Figure 1.

This is a dynamic sequence diagram that shows the interaction between modules for the use case of creating a new account.

4.2 User Schedules a Service

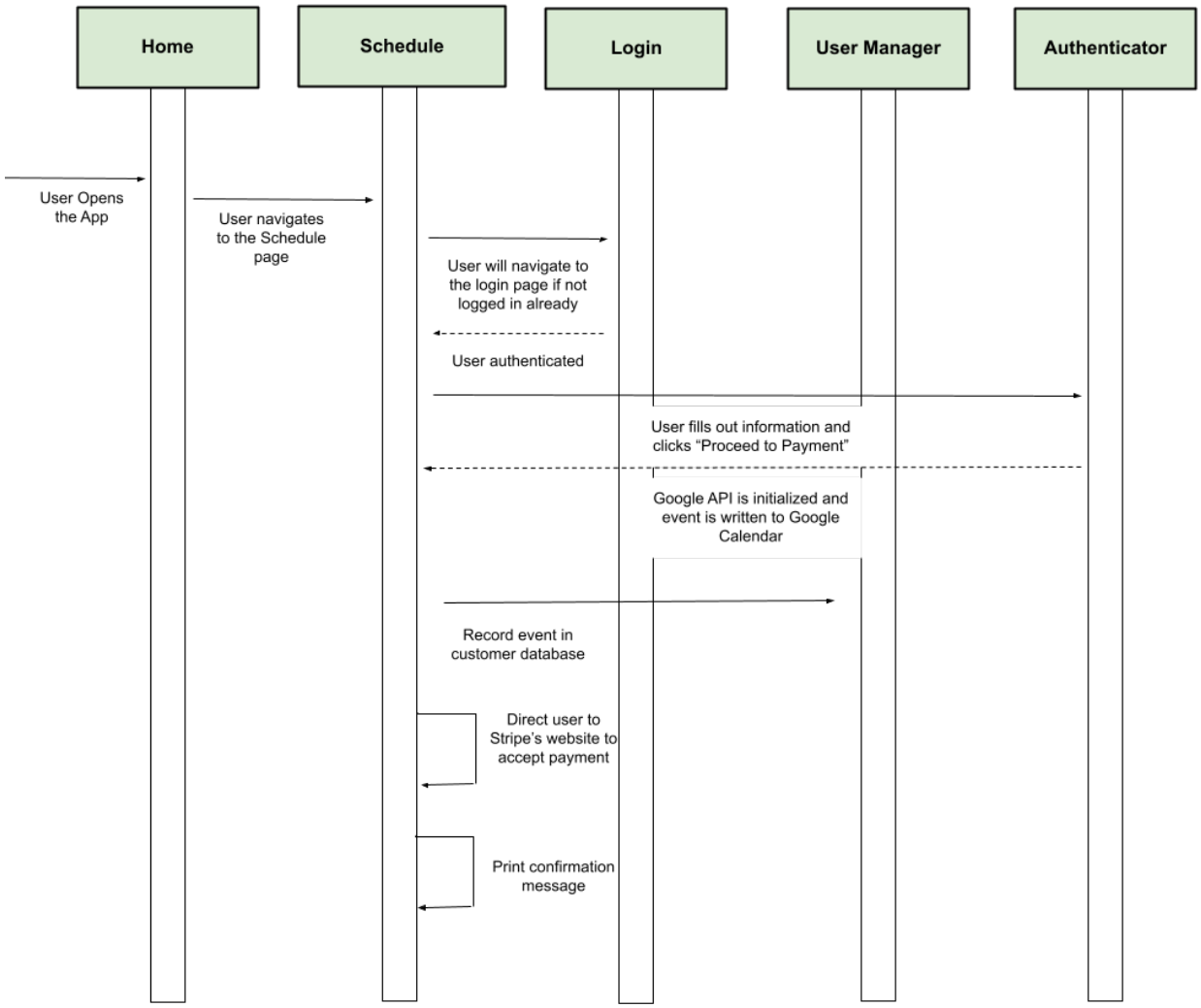

Figure 2.

This is a dynamic sequence diagram that shows the interaction between modules for the use case of scheduling a service.

4.3 Viewing Booked Services

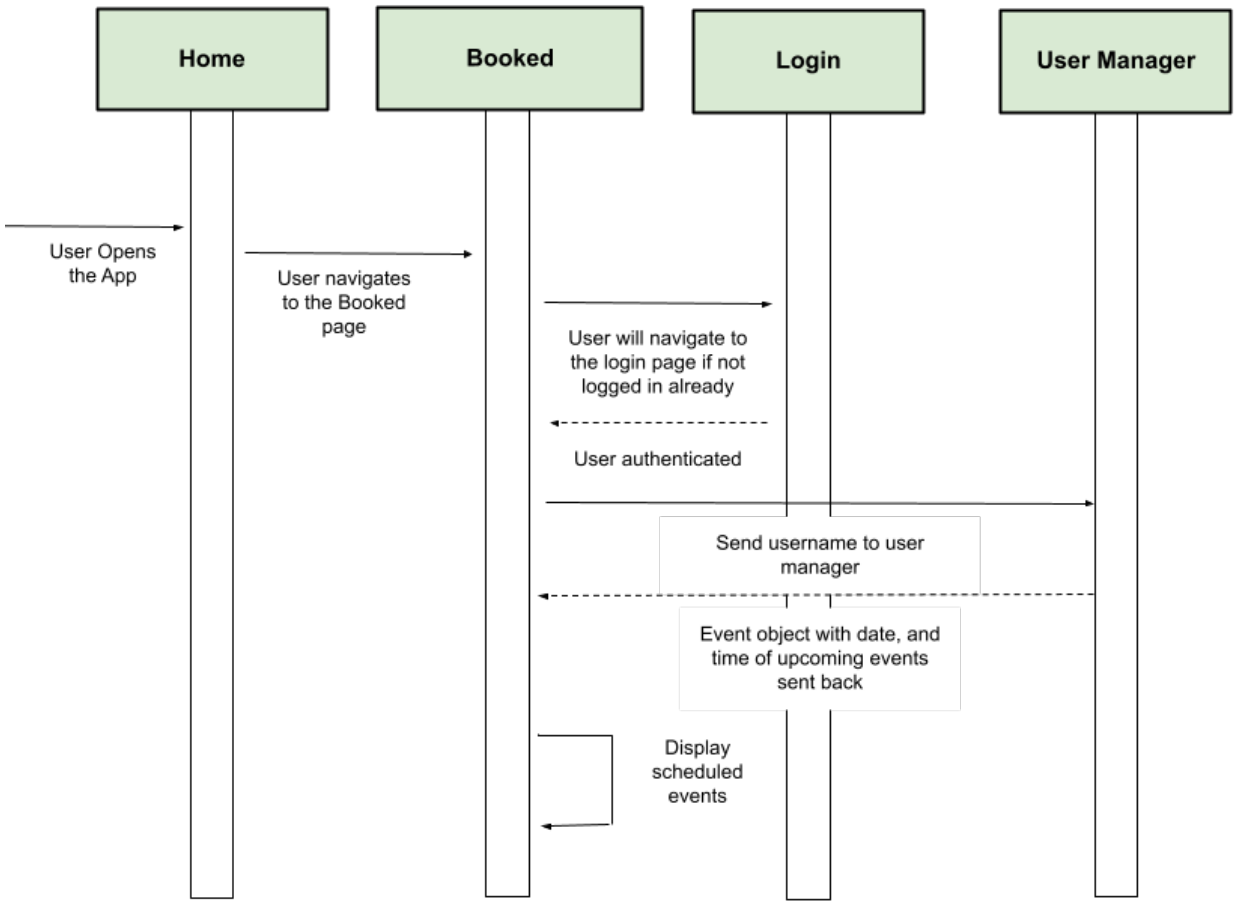

Figure 3.

This is a dynamic sequence diagram that shows the interaction between modules for the use case of viewing a booked service.

Installation Instructions

User

This program is currently in the prototype phase. Therefore, a user will not be able to view or download this program yet. Once deployed, the user will simply navigate to a website to operate this system.

Programmer

A programmer or business owner who wants to modify this system to fulfill their own purpose can download the repository from GitHub.

1. Navigate to https://github.com/apelky/AngelaWalksDogs
2. Click “code” then “download zip”

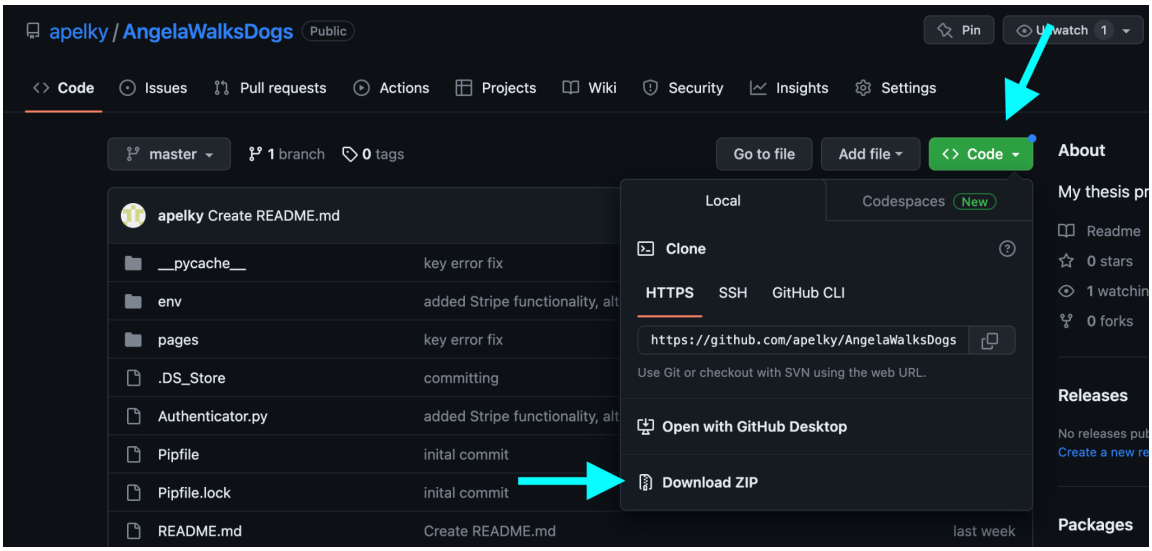

Figure 4.

Photo detailing how to download a GitHub repository.

Programmer Documentation

Setting Up Your Environment

Visual Studio Code

1. Download visual studio code by following these steps.
2. Follow along with the documentation to set up and get started.
3. Open the Angela Walks Dogs repository by clicking "open" on the Visual Studio Code home page.
4. Create a virtual environment using venv by following the steps listed here.
5. Once your environment has been activated, you can download the necessary packages within this environment. Navigate to the terminal within Visual Studio Code. Run the following command line prompts:
   a. `pip install streamlit`
   b. `pip install PyYAML`
   c. `pip install streamlit-authenticator`
   d. `pip install --upgrade google-api-python-client google-auth-httplib2 google-auth-oauthlib`
   e. `pip install Pillow`
   f. `pip install pysqlite3`
   g. `pip install web-browser`
6. Display line number for code by following these steps.

Opportunities to Personalize

1. home.py
   a. Modify information in " " between `lines 21-69` to customize your page
   b. To modify the image, navigate to the folder AngelaWalksDogs. Simply delete the photo "`home.jpg`", upload your own photo, and rename it to "`home.jpg`".
2. authenticator.py
   a. On `line 50`, replace `calendarId='contact.angelawalksdogs@gmail.com'` with your own gmail account.
   b. On `line 50` replace the return message with whatever you'd like.
   c. Before this will run, you must complete the following steps:
      i. Navigate to: https://console.cloud.google.com/projectselector2/apis/dashboard?supportedpurview=project&authuser=1

ii. Click “Create Project”
iii. Click “Enable APIs and Services”
iv. Search “Google Calendar”
v. Select the first option “Google Calendar API”
vi. Click “Enable”

d. Now, authorize your credentials by following the steps listed here

3. schedule.py
   a. On `lines 30-33`, replace with relevant time zones and Stripe checkout links
   b. Find out how to get started with Stripe here
   c. You may want to modify some parts of this script to match your specific needs. The components that you may want to modify are (1) duration (2) stripe links (3) the calendar you would like to embed (4) the kind of service the user will book (5) the various needed user information (6) Event stored in Google Calendar. Let us look at each of these in more detail.

(1) Duration of the service

```
# Helper function that asks the user how long they'd like their service to be.
def duration():
   duration = st.radio(
#-----------------------------------------------------------------------------------
# Modify prices and times to match specific needs
       "Select service length",
       ["20 Minutes - $15", "30 Minutes - $20", "60 Minutes - $30"],
# End modification
#-----------------------------------------------------------------------------------
       horizontal=True
   )
   return duration[:2]
```

Figure 5.

`st.radio` creates a menu for your user to select. It returns a string identical to the option the user chooses. This helper function returns the number of minutes the booking should take in the format of a string. If you would like to change this, modify `('20 Minutes - $15', '30 Minutes - $20', '60 Minutes - $30')`. Please note that this program will break if you do not format your changes correctly. A necessary requirement is that you have two numbers at the beginning of each sentence. For example, replacing `'20 Minutes - $15'` with `'Duration: 20 Minutes'` would break the program. As long as you have two integers as the first two characters of each string you will be golden.

(2) Payment Links

```
# Helper function that opens Stripe Page once button is clicked
def open_browser(duration):
#-----------------------------------------------------------------------------------
# Length of service paired with Stripe Checkout links
   if duration == '20':
```

```
        wb.open(STRIPE_CHECKOUT_20)
    elif duration == '30':
        wb.open(STRIPE_CHECKOUT_30)
    elif duration == '60':
        wb.open(STRIPE_CHECKOUT_60)
# End modification #-------------------------------------------------------------------
------------------
```

Figure 6.

Depending on the modifications made in (1), you will have to make modifications in (2). Change orange text to match the duration of your services in (1).

(3) Calendar you would like to embed

```
# embed calendar for viewing within my app
components.html(
    """
    <iframe
src="https://calendar.google.com/calendar/embed?src=contact.angelawalksdogs%40gmail.co
m&ctz=America%2FLos Angeles" style="border: 0" width="600" height="600"
frameborder="0" scrolling="no"></iframe>
    """,
    height=700,
)
```

Figure 7.

Follow the steps listed here to grab the correct html code. Once you've copied your code, replace `<iframe src="https://calendar.google.com/calendar/embed?src=contact.angelawalksdogs%40gmail.com&ctz=America%2FLos_Angeles" style="border: 0" width="600" height="600" frameborder="0" scrolling="no"></iframe>`. You can change the width and height of your calendar by modifying the numbers inside `width="600" height="600"`.

(4) Kind of service you'd like to book

```
#-----------------------------------------------------------------------------------
# Modify the type of service to fit your needs
        'What kind of service would you like to book?',
        ('Walk', 'Drop-In')
# End modification
#-----------------------------------------------------------------------------------
```

Figure 8.

Alter the words `'Walk'` and `'Drop-In'` to match the kinds of services that you offer. This could be

something like ‘Eyelash Extension’ ‘Eyebrow Wax’. Please note that you can add more options as well by simply adding a comma and another string. For example, ('Walk', 'Drop-In', 'Sitting').

(5) Needed user information

```
get_name = st.text_input(
#-----------------------------------------------------------------------------------------
# Replace with relevant information
   "Enter your pet's name"
   )
# DELETE if location not needed
   get_location = st.text_input(
   'Enter your address'
   )
# Replace with the prompt you'd like the user to provide
   get_notes = st.text_input(
   'Enter any access instructions I would need to get inside the home. For example:
the access code is xyz or I left the key under the mat!'
   )
# End modification
#-----------------------------------------------------------------------------------------
```

Figure 9.

If you are not in the dog business, some of these questions might not be relevant to you. For example, you might want to ask something other than “Enter your pet's name”. Let’s say that you want to know the person’s name instead. If that’s the case you could change it to “Enter your name”. get_name will then store whatever the user types in. If you do not need to know someone’s address, you can delete

```
get_location = st.text_input('Enter your address')
```

If you decide to delete this, please see (5). Finally, prompt the user with anything else you need to know about them. Replace 'Enter any access instructions I would need to get inside the home. For example: the access code is xyz or I left the key under the mat!' with your own prompt.

(6) Event stored in Google Calendar

```
# Helper function that builds the event dictionary in preparation for calling the
Google Calendar API.
   def schedule_builder(w_or_drop):
       r_day = schedule()
       r_time = time()

       # must convert it to a full datetime object before adding the duration
       datetime_obj = (datetime.datetime.combine(r_day, r_time))
       # add the duration to the start time to create an end time
```

```
        end_time = (datetime_obj + datetime.timedelta(minutes=int(r_duration)))
        # convert our start and end time to match Google Calendar requirements
        datetime_obj = datetime_obj.strftime('%Y-%m-%dT%H:%M:%S-07:00')
        end_time = end_time.strftime('%Y-%m-%dT%H:%M:%S-07:00')

        event['summary'] = w_or_drop+get_name
#-------------------------------------------------------------------------------------
# DELETE if deleted lines 127-130
        event['location'] = get_location
# End modification
#-------------------------------------------------------------------------------------
        event['description'] = get_notes
        event['start'] = {'dateTime': datetime_obj, 'timeZone':DEFAULT}
        event['end'] = {'dateTime': end_time, 'timeZone':DEFAULT}

        return event
```

Figure 10.

This is built for an event that will be scheduled in PST. If you are in a different time zone, please change `datetime_obj = datetime_obj.strftime('%Y-%m-%dT%H:%M:%S-07:00')` `end_time = end_time.strftime('%Y-%m-%dT%H:%M:%S-07:00')` 07:00 to your correct zone. You can find zone codes here.

Delete line `event['location'] = get_location` if you deleted `get_location = st.text_input('Enter your address')` in (5).

(7) Complete if modified `'Walk'` and `'Drop-In'` in (4)

```
#-------------------------------------------------------------------------------------
# Set option equal to corresponding changes on line 55
   if option == 'Walk':
        event = schedule_builder('Walk with ')
# End modification
#-------------------------------------------------------------------------------------
        run = st.button('Proceed to Payment')

        if run:
            open_browser(r_duration)
            st.write(main_auth(event))
            add_userdata(get_user,event)

#-------------------------------------------------------------------------------------
# Set option equal to corresponding changes on line 55
   if option == 'Drop-In':
        event = schedule_builder('Drop-In with ')
# End modification
```

Figure 11.

Replace `'Walk'` with the same exact string used in (4). Then, modify `'Walk with '` in the same manner. So, if you replaced `'Walk'` with `'Eyelash Extension'` it would be `'Eyelash Extension with '`. Do the same thing for `'Drop-In with '`. If you added another option at (4) then you just need to copy and paste and replace with the added string. So, based off my example in (4), I would include

```
if option == 'Sitting':
      event = schedule_builder('Sitting with ')
      run = st.button('Done')
      if run:
            confirmation = main_auth(event)
            st.success(confirmation)
            next_steps()
            add_userdata(get_user, event)
```

4. config.toml
    a. In this file, the Streamlit theme is curated. Alter color codes to change color scheme.
5. main.css
    a. Similar to 4, the font and overall page style can be tweaked to create a different look for the web application.

Limitations

Although this is an exciting new project, there are some limitations to consider.

Security

To engineer this dog walking app within time constraints and programmer expertise, security compromises had to be made. These security weaknesses are not insignificant and should be taken seriously. For this reason, this web application will stay in the prototype phase until the security of the program has been proven.

Login Information

This web application is responsible for handling, storing, and securing sensitive user data necessary for a login process. This data includes

1. Email
2. Username
3. Name
4. Password

Which is stored in a local yaml file. Although the passwords in this file are encrypted, the rest of the data can be easily extracted.

Service Booking

When booking a dog walking service, a user must relinquish

1. Address
2. Access instructions

This information is needed because the dog walker must be able to travel to the home and retrieve the pet. Although, this is obviously very sensitive information. As it is currently engineered, the address and access information are written into the Google Calendar event. Although this event is only visible to "contact.angelawalksdogs@gmail.com," Google Calendar is not engineered to hold sensitive information.

Usability

The usability of a web application is by far the most important feature for customer retention and satisfaction. As with any computer program, engineers must consider tradeoffs between usability, cost, and security. In the case of Angela Walks Dogs, it is less usable, but it is free. Therefore, it could be more usable in terms of the booking process and the payment process.

Booking Process

As it is currently engineered, the user is required to manually view available time slots via Angela's embedded Google Calendar. To increase functionality, remove the Google Calendar and instead only allow the user to select available time slots in the drop-down menu.

Payment Process

In "How to Build an App" Patel emphasizes the need to make it easy for people to pay [12]. Right now, the user is required to follow a link to another web page to finish the payment process. Instead, use the Stripe API to embed the payment process so that the user is not allowed to schedule a service without first paying for the service.

Conclusion

Angela Walks Dogs is a revolutionary project that combines the fields of computer science and entrepreneurship to create a digital business platform template through the use of the Python library Streamlit. Through rigorous research this thesis has defended the need for a new dog walking platform. Unlike the current systems Wag! and Rover, this innovative web application is strategically engineered to focus on the caregiver and their specific skill set. It provides a simplified structure that will allow customers to learn how to navigate the web application easily to promote customer lock-in. Furthermore, design decisions were made to ensure this web application is reproducible and individual. Therefore, any small business owner can follow the steps outlined in this thesis to build their own independent platform that is free to host and completely customizable. Computer science is an incredibly diverse and powerful tool that can help anyone if they are willing to try.

Appendix

Introduction to Streamlit

Install Stramlit [13]

- This walks through similar steps defined in programmer's documentation

Main Concepts

- streamlit run your_script.py [-- script args]
    - Interestingly, when you run your code it opens up your web app on web browser
- Anytime you need to update the app you simply save the source file. On the web page you can choose "Always rerun" to have it automatically update for you, so when you save it you don't even have to run it again it just updates.
- Widgets include sliders, buttons, and selectbox - every time a widget is moved or changed the script is rerun from top to bottom
    - Streamlit makes it easy to organize your widgets in a left panel sidebar with st.sidebar. Each element that's passed to st.sidebar is pinned to the left, allowing users to focus on the content in your app while still having access to UI controls.
    - st.columns lets you place widgets side-by-side, and st.expander lets you conserve space by hiding away large content.
- You can change the themes & color - dark mode, light mode, etc
- You can organize your app by page as well
    - In the folder containing your main script, create a new pages folder. Let's say your main script is named main_page.py.
    - Add new .py files in the pages folder to add more pages to your app.
    - Run streamlit run main_page.py as usual.
- Overarching summary
    - Streamlit apps are Python scripts that run from top to bottom
    - Every time a user opens a browser tab pointing to your app, the script is re-executed
    - As the script executes, Streamlit draws its output live in a browser
    - Scripts use the Streamlit cache to avoid recomputing expensive functions, so updates happen very fast
    - Every time a user interacts with a widget, your script is re-executed and the output value of that widget is set to the new value during that run.
    - Streamlit apps can contain multiple pages, which are defined in separate .py files in a pages folder.

Multi Page App

- Page labels in the sidebar UI are generated from filenames. They may differ from the page title set in st.set_page_config. Let's learn what constitutes a valid filename for a page, how pages are displayed in the sidebar, and how pages are sorted.

- Valid filenames for pages
    - Filenames are composed of four different parts:
        - A number — if the file is prefixed with a number.
        - A separator — could be _, -, space, or any combination thereof.
        - A label — which is everything up to, but not including, .py.
        - The extension — which is always .py.
- How pages are displayed in the sidebar
    - What is displayed in the sidebar is the label part of the filename:
        - If there's no label, Streamlit uses the number as the label.
        - In the UI, Streamlit beautifies the label by replacing _ with space.
- How pages are sorted in the sidebar
    - Sorting considers numbers in the filename to be actual numbers (integers):
        - Files that have a number appear before files without a number.
        - Files are sorted based on the number (if any), followed by the title (if any).
        - When files are sorted, Streamlit treats the number as an actual number rather than a string. So 03 is the same as 3.
- Pages share the same Python modules globally:

```
# page1.py
import foo
foo.hello = 123

# page2.py
import foo
st.write(foo.hello)  # If page1 already executed, this should write 123
```

Text Elements

Markdown

st.markdown(body, unsafe_allow_html=False)

- **text** - bold
- _text_ - italics
- Supports colors and emojis

Title

st.title(body, anchor=None), st.header(body, anchor=None), st.subheader(body, anchor=None), st.caption(body, unsafe_allow_html=False)

- Each document should have a title
- Body is displayed as markdown

Text

st.text(body) - Write fixed-width and preformatted text.

Input Widgets

Buttons

st.button(label, key=None, help=None, on_click=None, args=None, kwargs=None, *, type="secondary", disabled=False)

- Label can contain markdown
- Returns True if the button was clicked on the last run of the app, False otherwise.
- If you're using a button make sure it's the last nested element that you use because it is meant to be only clicked once, once it's clicked it will remember

Check Box

st.checkbox(label, value=False, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")

- Ex:

```
import streamlit as st
agree = st.checkbox('I agree')

if agree:
    st.write('Great!')
```

Select Box

st.selectbox(label, options, index=0, format_func=special_internal_function, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")

- Ex:

```
option = st.selectbox(
    'How would you like to be contacted?',
    ('Email', 'Home phone', 'Mobile phone'))

st.write('You selected:', option)
```

Text Input

st.text_input(label, value="", max_chars=None, key=None, type="default", help=None, autocomplete=None, on_change=None, args=None, kwargs=None, *, placeholder=None, disabled=False, label_visibility="visible")

Date Input

st.date_input(label, value=None, min_value=None, max_value=None, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")

Time Input

st.time_input(label, value=None, key=None, help=None, on_change=None, args=None, kwargs=None, *, disabled=False, label_visibility="visible")

- Ex:

```
t = st.time_input('Set an alarm for', datetime.time(8, 45))
st.write('Alarm is set for', t)
```

Media Elements

Image

st.image(image, caption=None, width=None, use_column_width=None, clamp=False, channels="RGB", output_format="auto")

Layouts and Containers

Sidebar

- Not only can you add interactivity to your app with widgets, you can organize them into a sidebar. Elements can be passed to st.sidebar using object notation and with notation.
- Ex:

  ```
  # Object notation
  st.sidebar.[element_name]
  # "with" notation
  with st.sidebar:
      st.[element_name]
  ```
- Each element is pinned to the left
- Resizable, can drag and drop the right border to resize it

Display Progress and Status

Success

st.success(body, *, icon=None)

Control Flow

Form

st.form(key, clear_on_submit=False)

- Create a form that batches elements together with a "Submit" button.
- A form is a container that visually groups other elements and widgets together, and contains a Submit button. When the form's Submit button is pressed, all widget values inside the form will be sent to Streamlit in a batch.
- To add elements to a form object, you can use "with" notation (preferred) or just call methods directly on the form. See examples below.
- Forms have a few constraints:
    - Every form must contain a st.form_submit_button.
    - st.button and st.download_button cannot be added to a form.
    - Forms can appear anywhere in your app (sidebar, columns, etc), but they cannot be embedded inside other forms.

Form Submit Button

st.form_submit_button(label="Submit", help=None, on_click=None, args=None, kwargs=None, *, type="secondary", disabled=False)

- Display a form submit button.
- When this button is clicked, all widget values inside the form will be sent to Streamlit in a batch.
- Every form must have a form_submit_button. A form_submit_button cannot exist outside a form.

Utilities

Page Configuration
st.set_page_config(page_title=None, page_icon=None, layout="centered", initial_sidebar_state="auto", menu_items=None)

- Ex:

```
st.set_page_config(
    page_title="Ex-stream-ly Cool App",
    page_icon="🧊",
    layout="wide",
    initial_sidebar_state="expanded",
    menu_items={
        'Get Help': 'https://www.extremelycoolapp.com/help',
        'Report a bug': "https://www.extremelycoolapp.com/bug",
        'About': "# This is a header. This is an *extremely* cool app!"
    }
)
```

Session State

Session State is a way to share variables between reruns, for each user session.

- Bit like a dictionary

Components

Components
st.components.v1.iframe(src, width=None, height=None, scrolling=False)

- Load a remote URL in an iframe.
- This is used for situations where you want to include an entire page within a Streamlit app
- This will be useful with my integrations such as Stripe and Calendar services

Introduction to APIs

APIs for Beginners
Video provides an excellent introduction to APIs

Google Calendar API

Overview of the Google Calendar API, how to set it up, and how to use it.

Bibliography

[1] *Pet ownership statistics [2022]: U.S pet population* (2023) *Spots.com*. Available at: https://spots.com/pet-ownership-statistics/#oregon (Accessed: 08 May 2023).
[2] May 8, 2023 from https://www.census.gov/quickfacts/fact/table/eugenecityoregon/PST045222
[3] Pet ownership statistics [2022]: U.S pet population. (April 2023). Retrieved May 8, 2023 from https://spots.com/pet-ownership-statistics/#oregon
[4] Book dog boarding, dog walking and more. Retrieved May 8, 2023 from https://www.rover.com/
[5] Nicole Cosgrove. 2022. Wag! dog walking & sitter app review 2023: Pros, Cons & Verdict. (October 2022). Retrieved May 8, 2023 from https://www.hepper.com/wag-dog-walker-sitter-app-review/
[6] Kimberly Alt. 2023. Rover pet sitting reviews: Are they safe? background check, prices, Promo Code, and more. (May 2023). Retrieved May 8, 2023 from https://www.caninejournal.com/rover-dog-sitting-reviews/
[7] Lauren Schwahn, Elizabeth Ayoola. Rover vs. WAG: Which app is better for making money? Retrieved May 8, 2023 from https://www.nerdwallet.com/article/finance/rover-vs-wag
[8] Difference between web application and website. (January 2022). Retrieved May 8, 2023 from https://www.geeksforgeeks.org/difference-between-web-application-and-website/
[9] Muhammad Aljukhadar and Sylvain Senecal. 2009. How the website usability elements impact performance. *Lecture Notes in Business Information Processing* (2009), 113–130. DOI:http://dx.doi.org/10.1007/978-3-642-03132-8_10
[10] Nurainun. 2019. Customer loyalty and customer relationship management. *Proceedings of the 20th Malaysia Indonesia International Conference on Economics, Management and Accounting* (2019). DOI:http://dx.doi.org/10.5220/0010520300002900
[11] 1998. *IEEE Guide for Information technology--: System definition-- concept of Operations (ConOps) document*, New York, N.Y, USA : Institute of Electrical and Electronics Engineers.
[12] Patel K. How to Build an App. Marketing Magazine. 2009;114(19):13. Accessed May 8, 2023. https://search.ebscohost.com/login.aspx?direct=true&db=f5h&AN=44788181&site=ehost-live&scope=site