PetSTAR

Authors: Rebecca Manley, Brandon Wei
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of detecting and tracking animal movement using a Raspberry Pi and camera and displaying this information through a web application. The system also allows a user to specify areas where their pet is not allowed and notifies them when the animal enters the area. Our project is a competitively priced hardware solution with functionality beyond traditional pet monitors.**

***Index Terms*—Axios, Computer Vision, Django, Forbidden Zones, OpenCV, Raspberry Pi, React, Web Application**

1 INTRODUCTION

Pet cameras are becoming increasingly popular as the technology becomes more accessible and people find themselves spending more time away from their home and pets. These systems provide pet owners additional peace of mind and offer insight about their pet's behavior even when no one is around. At their core, traditional approaches to pet monitoring are all quite similar - a camera which watches whenever the owner is away and transmits this feed to their personal device. However, this means that the pet owner would need to be actively watching (or spend time reviewing footage later) in order to get any idea of what their pet has been up to.

In this project, we set out to make the task of pet monitoring more convenient in two main ways: providing live notifications when an animal goes somewhere it shouldn't and reporting summarized pet activity via a heat map. This is accomplished by incorporating computer vision algorithms to detect and track animal movement and a web application to allow user input and convey important information. We also aimed to keep our final product competitive with the current market by maintaining a low cost of components. It is our belief that this system could prove valuable in any home with pets.

2 USE-CASE REQUIREMENTS

The use case for our project is, broadly, any home that has one or more free-range pets. The first general category of our use case requirements relates to the overall functionality that we want to achieve with our system. Since we will alert the user when a pet goes somewhere it shouldn't, we want to minimize unnecessary disruptions to the user by having a low false positive rate of $< 10\%$. We also want to make sure that these reports reach the user quickly so that they are aware of any potentially dangerous situations with their pet(s). Counting from the time that the animal enters the forbidden area, we aim to be able to detect and notify the user within 10 seconds. Another core aspect of our functionality is the summarizing activity logs that we will provide for each animal. Since these logs are only useful to the user if they are accurate, our goal is to have logs which accurately reflect the movement of the animal(s) $> 90\%$ of the time.

The other broad category of our use case requirements relates to accessibility. Since there is an enormous diversity in homes that have pets, it is important to us that we keep our project as inclusive as possible. For our finished project, it is our goal that the average user would be able to set up with system in ≤ 5 minutes, with instructions. We also want the web application to be intuitive and user friendly, with $> 90\%$ of users able to complete the core tasks (select forbidden zone, upload pet images, request activity logs, etc.) with little or no additional instruction. Lastly, we want to keep our system as affordable as possible. We aim to keep the overall cost to $\leq$ \$100 as our research has indicated that \$100 is roughly the starting price for other pet monitoring systems with features beyond just an app-connected camera.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

Our physical system is depicted in Figure 1. It consists of a Raspberry Pi and camera module. It also requires the power adapter and a micro SD card to load the RPi operating system.

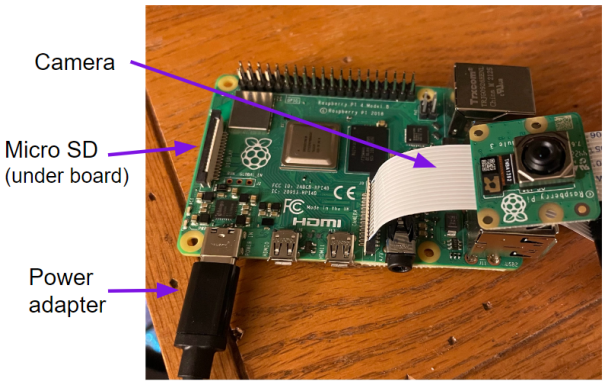

Figure 1: The physical hardware for our project

The RPi runs the computer vision algorithms which analyze the incoming video to identify and track the pets. It

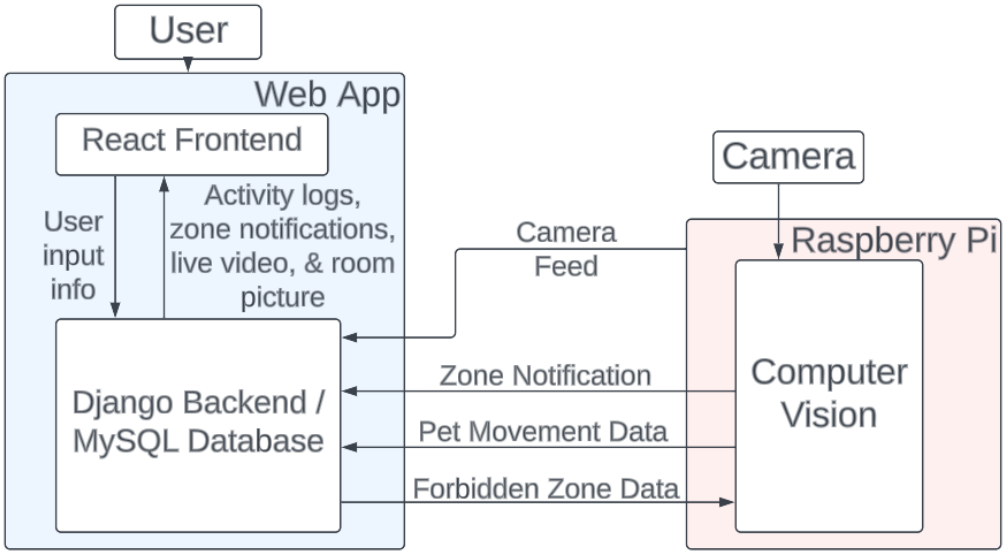

Figure 2: Block diagram for the overall system

then communicates with our web application by connecting to the user's in-home wireless internet.

Our architecture has changed fairly significantly from what we discussed in the design report. Most notably, all elements of machine learning have had to remain separate due to personal setbacks. The re-scoped project relies only on the CV motion detection and tracking to monitor any animals in the room, and still communicates with the web application as planned. As such, our project no longer aims to differentiate between multiple animals in the home. The user may only specify one forbidden zone which applies for all pets, and activity logs will reflect the movements of all animals. It also no longer requires users to upload images of their pets in the web application. Note that this feature is still present in the web application, for demonstration purposes, but does not do anything. Another, more minor change to the project is that the speaker feature has been removed entirely. Lastly, all communications with the web app go through the back end, i.e. nothing goes directly to or from the front end. This is to ensure that all necessary data that we need to store in the back end has a chance to get logged, and generally simplify communication protocols. See Figure 2 for the updated high level view of the system.

The web app is where the user will be able do all the relevant tasks and receive notifications for monitoring his/her pet(s). First time users will be prompted to pair with their camera, then be able to view live video feed to position their camera. When they're happy with the positioning they need to press a button to save a template image of the room. After this (or immediately, for returning users) the user will end up on the PetSTAR dashboard. The 'forbidden zone' task will allow users to select areas where they do not want their pet to go in the room picture taken previously by clicking on grid squares overlayed on the image. After finishing this process, the user will receive notifications on the web app whenever a pet has entered a forbidden zone. The 'activity logs' task lets users view pet activity logs through a heat map that shows the movement of the pet in the room over a long period of time. The user can clear these logs at any time, and it is recommended they do so before each extended use of the camera. The 'new pet' tasks allows a user to enter the name and picture(s) of each pet. The 'live video' task lets users view a live video feed of the room, which is similar to what traditional cameras offer. They can also use this to reposition the camera and re-select a room photo whenever they would like. See Figure 3 illustrates the flow of how the user will interact with the front end app, and Figure 4 shows the dashboard appearance.

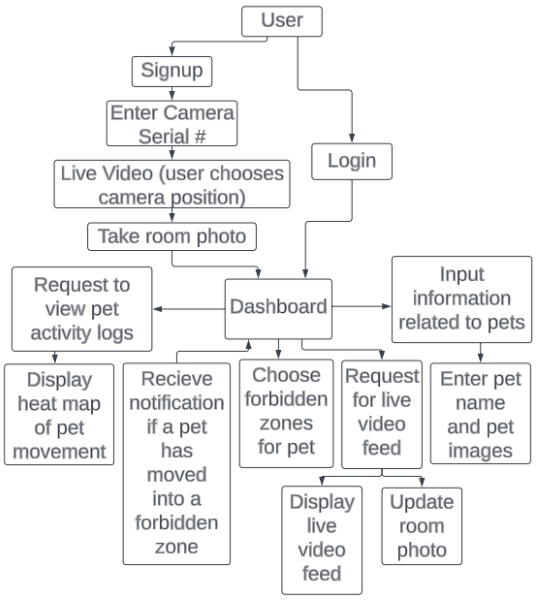

Figure 3: Web app user flow diagram

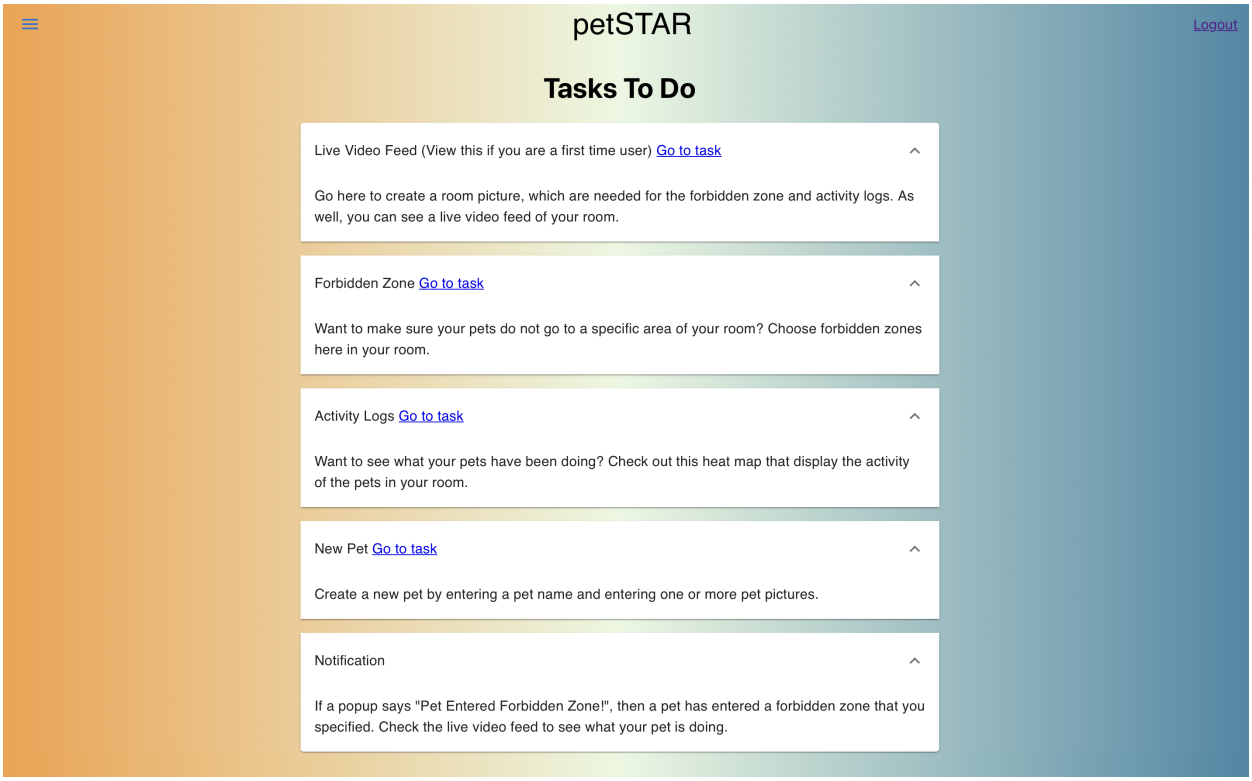

Figure 4: Final appearance of web application dashboard

The computer vision aspect of the project will be based mostly in the OpenCV library. It aims to detect motion to see when an animal enters the frame. Once motion is noticed it will assign a tracker to that animal in order to follow it as it moves through the environment. This allows us to not lose sight if an animal goes still. Using the forbidden zone data communicated from the web app, it will check for significant overlap between the position of the animals and any forbidden squares, raising a notification flag if needed.

Communications between the RPi and web app will be done via an internet connection. The web app will communicate user-input data, i.e. the user defined forbidden zones. The RPi communicates back with activity data and a live camera feed that can be displayed to the user as well as pushing a notification whenever an animal enters a forbidden area.

4 DESIGN REQUIREMENTS

The first use case requirement we laid out pertained to the speed of the overall system: Users must be notified quickly (in < 10 seconds) when an animal has entered a forbidden zone. In order to achieve this, we want our computer vision to be able to flag when an animal has entered a forbidden zone within 1 second of it having entered. The remaining overhead time (9 seconds) is mostly a remnant from anticipating the time it would take to use machine learning to identify the animal. It also provides space for the small amount of time required for the notification to travel from the RPi to the frontend of the web app.

The next use case requirements relate to the accuracy of our system. In order to avoid false positive notifications to the user, we want our computer vision to track the animals within 1 foot of their actual position. Note that we are considering the animal's position to be its center of mass. We are using a grid based system to specify forbidden zones, with the expectation that the user will place the camera far enough back to capture most of the room. This means that a grid box should be around 1 foot or more in our intended use case, so we should always be within one grid box in any direction of an animal's true position. For our average use case, we expect the forbidden zone to be off of the ground, such as a shelf or a table. In these cases, it would be very unlikely that the animal could be just outside of the zone for any prolonged period of time (as this would imply hovering), which could result in a false positive. In edge cases where the forbidden zone is closer to somewhere the pet could legally occupy, false positives could be more likely depending on how the pet behaves. It may be the case that the animal has no reason to approach this area other than to enter the forbidden zone, in which case false positives should remain low. If the animal commonly moves near the forbidden zone in the course of normal activities, though, then false positives may be high. This is something that the user will be informed of.

Our ability to maintain $> 90\%$ accurate pet activity logs also depends on this tracking accuracy. When the animal is further from the camera, 1 foot is smaller and less

likely to span multiple boxes. When it is closer, the animal itself will appear bigger and thus occupy more boxes. The logs consider a broader portion of the space occupied by the animal (than just its single point center of mass), so up to 1ft of inaccuracy would still count most of the mass as being in appropriate squares. As such, the activity logs would still express activity in the correct area. Another important factor in our log accuracy is how quickly we can identify when an animal has entered the frame. Delays in noticing a new animal will lead to data being omitted from the activity log. We expect that this time will vary based on the speed that the animal is moving, but want an upper bound of 5 seconds for all cases. So long as the animal spends at least 50 seconds in front of the camera before leaving frame again (on average), this up to 5 seconds of omitted data will constitute less than 10% of the overall data.

Lastly, in order to maintain a low user cost, we want to use low cost hardware options. We want an overall system cost of $100 or less to make our option competitive with the current pet monitor market. We are achieving this using a Raspberry Pi (with a baseline price of $35 pre-pandemic, $45 post-pandemic) as our primary hardware in order to maintain a low hardware cost.

5 DESIGN TRADE STUDIES

5.1 Web App Architecture

Frontend

There are a few choices for the frontend including using plain JavaScript/TypeScript, Bootstrap, Angular, and React. Looking at these choices, React and Angular are the best at improving user experience and user interface. We chose React as the frontend of the product as the team member working on the Web App has prior experience with in other classes and React includes a library called Material UI that can be used to create components such as a sidebar and drop-downs on the Web App that help users navigate the website easier. As well, React is one of the most popular libraries used as a frontend to the web application, so documentation would be easy to find and understand for any problems that occur while developing the frontend.

Backend

For the backend, a few popular options include Node.js and Django. Both frameworks would have supported what the Web App needs to do in terms of functionality, but Django seemed to be the better option as the team member working on the Web App has prior experience with this framework in other classes and testing/integration with the CV would be easier as both Django and CV are in Python. One consideration was how to deploy the backend as it would be needed to integrate the Web App and CV, and the team member working on the Web App had prior experience with deploying using Django while having no relevant experience with Node.js.

Requests

Looking at communication between the frontend and backend, the overwhelming consensus was to use Axios and the Django REST framework for a React frontend and Django backend. Axios was used for passing data between frontend and backend while the Django REST framework parsed data from the Axios calls to the database stored at the backend. Other options for request calls include fetch and XMLHTTPRequest, but Axios was the better option as data passed was the actual object versus the string version of the object. As well, Axios has built-in CSRF protection, which can be useful when dealing with the ethical issues specified later in the report. It is possible not to use a REST framework when passing data from the request call to the database, but using the Django REST framework supported the validation of request data before being stored in the database.

Authentication

One design idea for authentication was to use Google OAuth 2.0. The team member working on the Web App had prior experience using this authentication process for project using Django only, and Google OAuth 2.0 allowed users to not have to create another login and ensured that users create accounts with not weak passwords. As well, this authentication process is a standard way of authentication that can be seen on many websites as it is usually incorporated in one of the single sign-on options on a website. As well, there has been an updated way to implement Google OAuth 2.0 authentication using Google Identity Services SDK, which means that the authentication process is up to date but also creates a lack of documentation for implementing this authentication process. Another design option for authentication is JWT authentication[2]. Unlike Google OAuth 2.0, this authentication process does not have a lack of documentation for authentication for a full stack web application such as this product. Hence, it would be easier to implement for this product. Similar to Google OAuth 2.0, JWT authentication uses access tokens to authenticating users, which means malicious users cannot easily access pages that are guarded by the access token. But, there are not many restrictions in terms of what users can enter for username and password, which means that they may be easily guessable by malicious users. This would lead to bad outcomes for those victims due to the ethical issues specified later in the report. Overall, we beleve JWT authentication is the authentication process to use for the Web App due to having documentation for full stack web applications and using access tokens, similar to Google OAuth 2.0, for authentication.

5.2 User choosing forbidden zones

We had to think about how to implement a user choosing forbidden zones for an image of a room that the pet will be in. Specifically, we looked at the type of image displayed (2D or 3D image) and how we wanted to partition the forbidden zones (free forming zones or grid system).

Option 1: User choosing forbidden zones on a 3D Image

A benefit to allowing users choose forbidden zones on a 3D image is that users are able to be very specific in how they define these zones. For example, they are able to choose a forbidden zones in front of a box or behind a box, which is not possible with a 2D image. But, we believe they are many issues with implementing this option. Specifically, it is significantly harder to get a 3D model of a room just from the camera feed rather than retrieving a 2D image of the room. There would be more data needed to be stored in the database for a 3D image compared to a 2D image. In terms of user experience, it may be harder for users to navigate a 3D model of the room and choose forbidden zones compared to clicking and/or dragging on a 2D image to create forbidden zones.

Option 2: User choosing forbidden zones on a 2D Image using free forming zones

The advantage to free forming zones compared to a grid system is that the user is able to create finer forbidden zones as they can click and drag to create multiple zones which can fully encapsulate everything in the room that is forbidden without including parts of the room that is not forbidden for the pet to be in. As well, we believe a user will be less confused when trying to create these free foaming forbidden zones on a 2D image compared to navigating a 3D image to create forbidden zones. The main issues with this option is that storing this data into the database and giving this data for the computer vision on the Raspberry Pi to use is complex compared to the more simplistic data structure offered by using a grid system. As well, there is no sense of depth in the 2D image, so choosing in front of a box versus behind a box is not possible.

Option 3: User choosing forbidden zones on a 2D Image using a grid system

We believe this is the best option to implement as there is a simple data structure, specifically an array like structure, that can be used to store data into the database and also passed along to the Raspberry Pi. As well, this method should be not confusing for users as users will click on the grid squares to choose forbidden zones on the 2D image. But, they are still consequences such as this is the worst option out of the three in terms of choosing specific forbidden zones on an image. For example, the grid square may mostly include the forbidden zone that the user intended to create, but it will more than likely includes a few parts that the user did not want to include into the forbidden zone. To address this, we will make sure to get user feedback on grid square sizes so that users are satisfied with the forbidden zones they are creating on the image. Overall, we believe this option makes the back end of the project much more simpler while not having significant impact on the front end of the project. See Figure 5 for an example of our forbidden zone grid system.

5.3 Displaying of Pet Activity Logs

We thought about how we wanted to display pet activity to logs for users when they request them, specifically if we wanted to display a short vs long time frame and specific vs generalized pet positions.

Option 1: Time-sensitive graph

A time-sensitive graph displays a short time period of the pet's movement, but a user will be able to scroll back and forth in time to see the exact position of the pet in the room at a specific time. The benefits to this option is that if a user has in mind what time they want to see what his/her pet is doing or the user will only be gone for a short amount of time, then they can check this graph to get precise locations of where the pet has been in a specific time frame compared to a heat map. A consequence is that the user cannot get the overall picture of where the pet has been due to the short time frame that is offered by the graph, and the user will have to be more interactive with the graph compared to a heat map to understand what the pet has been doing.

Option 2: Heat map

A heat map will display locations of where the pet has been over a long time period using different colors that represent the spectrum between low amount of activity in a place of the room to a high amount of activity. The benefits to a heat map is that a user will receive information quickly about the pet's activity as the heat map is not interactive, and the user can get overall statistics on a pet's activity over a longer time frame such as a whole day. The consequences is that the user will not know where a pet is located at a specific time, which may be more of an interest to the user. Though both options seems to benefit the user in different ways, we believe this is the best option as it has a simpler implementation, and it is faster and easier for the user to understand the pet activity information. See Figure 6 for an example of what our heat map implementation looks like.

5.4 Motion Detection

Two methods of motion detection that we observed were to:

1. Look for pixel differences versus a canonical empty frame (usually the first frame)[10]
2. Look for pixel differences between each consecutive pair of frames[6]

The benefit to the first approach is that it very clearly and solidly identifies anything which is not in the canonical frame. It is also not vulnerable to losing the animal if it stops moving (so long as said animal was not in that exact position in the canonical frame). However, the trade off is that it would be incredibly sensitive to changes in lighting or position of the camera. Anything which makes the current view look non-negligibly different from the canonical frame will cause the whole frame to be flagged, and generally is not recoverable without resetting the canonical frame.

The benefit to the second approach is pretty much the opposite of the first approach. Option 2 offers protection against non negligible changes in the camera scene, as this

Figure 5: Final appearance of web application forbidden zone interface

Figure 6: Final appearance of web application activity logs

difference will only register for however long it takes the camera to adjust to its new position/lighting. However, the downsides are that we will lose sight of the animal if it stops moving, and also the detected motion tends to focus on the outer edges of the animal and not necessarily encapsulate it entirely.

For our project we have gone mostly with the second approach, with some slight modifications. Specifically, to balance with some of the benefits in approach 1, we reset our comparison frame every 10 frames rather than every frame. This makes the perceived motion a bit larger and easier to encapsulate with a single bounding box. It also grants us the benefit of fairly high insensitivity to position/lighting, which is important given the risk of the animal bumping the camera, or the camera being used over a prolonged period with varying lighting. Note that this system is still vulnerable to when the animal stops moving, which is why we compensate by associating a tracker with each detected motion.

5.5 Hardware

One primary trade study that we focused on in the design report is which hardware platform we want to base our project around. The two options we'd been considering are a Raspberry Pi 4 (RPi) or and NVIDIA Jetson Nano. In short, we feel it is a tradeoff between performance (with the Jetson) and cost plus ease of use (with the RPi). Jetsons are specialized to deal with graphical processing, which we will be incorporating a lot of between the vision and ML aspects of our project, whereas an RPi is not. Implementing our system using a Jetson would certainly give better performance in terms of the frame rate that we're able to process. However, a Jetson is more expensive than an RPi, especially in the current market, as summarized in Table 1 (note that this assumes the cost for a model with 2GB memory).

When planning to include the machine learning, we were intending to use the more expensive and more computationally friendly Jetson once testing showed that it would be infeasible to use the RPi. However once the ML seemed likely to be separated from the project, we changed back to using the RPi as we had originally hoped to do - without the ML, the RPi was computationally sufficient for the CV code alone. This allowed us to keep our costs lower.

6 SYSTEM IMPLEMENTATION

6.1 Hardware

Our project is built using a Raspberry Pi 4 Model B with 4GB of RAM. The camera is a Raspberry Pi Camera Module 3, connected to the built-in camera interface on the board.

Table 1: Hardware Cost Comparison

Device	Pre-Pandemic	Current
Raspberry Pi 4	$35	$45
Jetson Nano	$59	$150

6.2 Web System

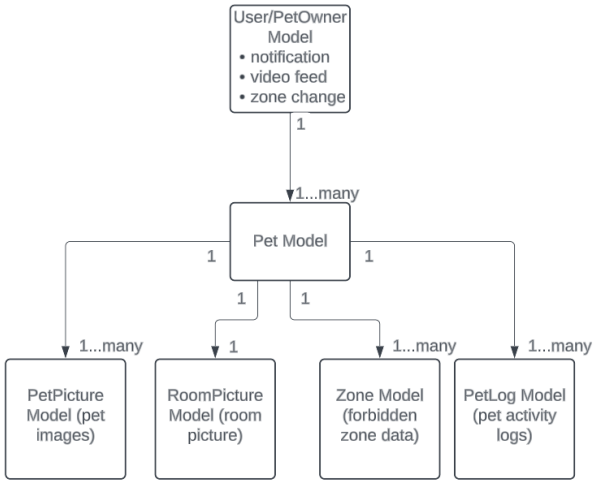

Figure 7: Data structure used in the back end server

The front end will be developed using React, which will allow us to create interactive tasks that should enhance user experience. Before interacting with these tasks, the user will login using JWT authentication. The back end will be developed using Django, which will store important data from user input on the front end. See Figure 7 for a visualization of how data is stored in the back end. From the design report to the final report, the method for storing pet related data has changed. From authentication, a User object is created for each pet owner that contains the username and password that the user inputs. Further, a PetOwner object, that stores data such as live video feed, if the forbidden zone has changed, and notification alerts, is created that is attached to that User object. While a user creates a room picture, a RoomPicture object is constructed that contains the room picture image, and this object has one to one relationship with a PetOwner object. If a new pet is input into the Web App, then a Pet object is created. As a pet owner can have multiple pets, then the Pet object has a many to one relationship with the PetOwner object. As multiple pet pictures can be given for each pet, then multiple PetPicture objects can be create for each Pet object. When a user does the forbidden zone task, each zone is stored as a Zone object where the fields are the position in the x direction, position in the y direction, and if it was marked forbidden or not by the user. Lastly for the activity log task, the data stored for each zone in the PetLog object is similar to the forbidden zone task except the amount of movement, which is determined by the amount of frames in the camera, is tracked instead of if the zone is forbidden or not. Both the Zone object and PetLog object have many to one relationships with the PetOwner object.

To connect the front end and the back end, a library called Axios will be used to call GET and POST requests to send data between the front end and the back end. As well, a toolkit called Django REST Framework will be used in between the Axios calls and the back end that will make storing data into the MySQL database and sending data from the MySQL database much easier. This web application will be deployed using Netlify for the frontend and Apache and Amazon EC2 for the backend.

While creating the Web App, multiple changes had to be made. To start, we moved from using Google OAuth 2.0 to JWT authentication due to lack of documentation for full stack web development using React and Django such as this product. Given that Google OAuth 2.0 was updated to use Google Identity Services SDK recently, there was lack of documentation on how to us the new update. So, we moved to using JWT authentication, which is similar to Google OAuth 2.0 as it uses access tokens for user authentication. Next, we switched the library we used from heatmap.js to jsheatmap. The former library seemed to be only compatible with JavaScript, which led to problems when implementing a heat map in TypeScript. Hence, we moved to jsheatmap, which was TypeScript compatible and had better documentation on how to implement a heat map. The drawback was that it seemed the heatmap.js displayed more visually appealing heat maps, but both libraries showed what is by definition a heat map, which means the user should not lose any information related to activity logs.

We changed how the room picture was taken and how the live video feed was displayed. For the room picture, we originally had the CV capture a frame of the video feed, which was then sent and displayed on the Web App. Though this was for testing to make sure we can create a room picture, we figured out this would be an inconvenience for users to do. Hence, we moved this operation onto the Web App. Specifically, the user is now able to add a room picture by clicking on a button on the live video feed page. For the live video tasks, we originally created a functional but very slow live video feed but sending a picture of the room and displaying the room every one second. As the page need to reload the image, this would create flickering on the page, which would ruin user experience. Hence, we found a StreamingHTTPResponse[5] object in Django, which is used in cases where the data being displayed is constantly changing. As this fits the live video feed description perfectly, we used this Response object type, and

this lead to an increase in frame rate of the live video feed on the frontend.

The most important lesson while creating the Web App was to constantly test each task I implemented. To do this, I used the Python requests package, which simulated calls between the CV and Web App and the Camera object from the OpenCV package, which simulated the CV. Hence, I was able to test every task on the Web App such as forbidden zones, pet activity logs, live video feed, and notifications, without the full implementation from the CV. Without this testing, it would have been extremely difficult to finish all the tasks in the development time given for the product.

6.3 Computer Vision Algorithm

The overall flow of the computer vision algorithm is visualized in Figure 8.

Each movement (which we assume to be a pet) is represented by a few pieces of information: a bounding box, a unique identifier, and an OpenCV tracker object.

Taking in the raw camera feed, obtained via Raspberry Pi's builtin picamera2 module, we convert each frame to a numpy array. For any pre-existing 'pets' that we are aware of from the previous frame, we feed the current frame to the OpenCV tracker so it may update itself. We process the frame for analysis by casting it to grayscale and blurring it a bit to smooth out noise. To detect new motion we then subtract it with a previous frame. Note that in our case we update the prev frame every 10th frame. We threshold the absolute value of this difference so that areas of significant difference (>25) become white and everything else becomes black. These white shapes mark our motion. We locate where these shapes are in the frame using OpenCV's builtin contour detection and get a bounding box for each motion. This list of new bounding boxes is compared against any bounding boxes we were already aware of, and if any significantly overlap with pre-existing 'pets' then they are merged into one bounding box - we assume that all movement in a nearby area comes from the same animal. If a motion bounding box doesn't correspond to any previous pets, we assume it is a new animal and assign it a unique ID and tracker.

Once this process is done, we go through all of the 'pets' that we have newly identified or have left over from previous frames. Since moving closer together may have brought them into collision, we check for any overlap and merge bboxes which conflict. Note that the original intention was that bboxes which had been verified by the ML would be assumed final, and so would not be merged or otherwise edited. Without the ML support, we simply let nearby pets merge if they are very close. They generally separate when they move apart again.

The result of these operations gives us our finalized list of pets for that frame. Forbidden zone collision is determined by seeing whether the center of any bounding boxes falls within a forbidden grid square. When measuring for the activity logs, we consider a rectangle with half the width and height of the overall bounding box, centered in the same spot as the bounding box. Any grid squares that this inner, smaller rectangle intersect with are marked as having activity. This lessens the over-representation in activity should the animal come close to the camera (such as using the whole bounding box). However it does so without overly reducing the space the animal is shown to have occupied (such as using only the center).

6.4 Overall System

Figure 9 at the end of this document shows our overall system block diagram, as well as where each part is coming from. To integrate all the components of the system, we will be using Python requests to send and receive data between the CV and Web App as CV uses Python, which means that we can use Python requests to create GET and POST requests to send data between the Web App and RPi. The Django REST framework is used to take data from these requests, validate the data, and store the data into the database. While integrating, we had to consider what data we needed to send between the major systems for each user task.

For forbidden zone tasks, the Web App sends an array containing a tuples of coordinates of each zone that was marked as forbidden by the user on the forbidden zone task to the RPi. The RPi looks if the zone change field stored in the PetOwner object is set to true, which is when the RPi takes forbidden zone data.

For activity logs task, the RPi tracks the amount of frames the pet has spent in each zone of the room picture. Periodically, this data is sent in an array that contains zone data from each zone in the order from top left to bottom right of the room picture. The Web App will parse this data, and update the heat map given this data. The Web App will keep count of the video frames spent in a certain zone of the room, so the RPi will reset the count of each zone after sending data to the Web App.

For the live video feed, the RPi will parse each frame from the camera into byte arrays, which is then encoded into base64. This is done so that none of the data sent through the Python request is lost. The Web App decoded the data from base64, creates an image using Pillow, a Python imaging library, and stores the image into the database. Then for the live video feed, the StreamingHTTPResponse object specified earlier is used to display the live video feed to the user.

For any notification triggered by the pet moving into the forbidden zone, the RPi calls a Python requests that sets the notification field in the PetOwner object to true. The frontend checks for this field and will display a popup displaying that a pet has entered a forbidden zone if that field is set to true.

While sending data related to forbidden zones and activity logs, one thing we had to debug was the coordinates sent between the Web App and RPI. We standardized the grid as the x coordinate changes in the horizontal direction while the y coordinate changes in the vertical direction.

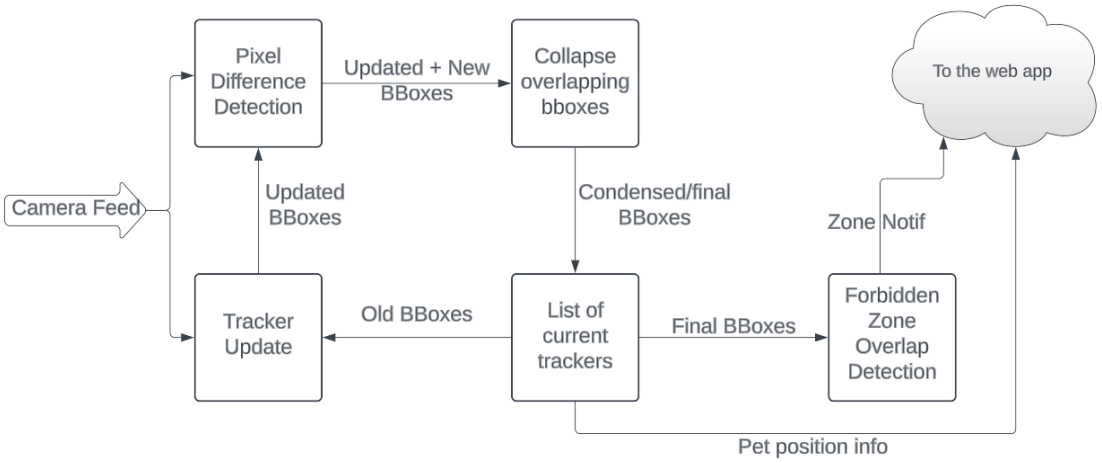

Figure 8: Layout of the Computer vision algorithms

This may create confusion as another common convention when using grids is to use row, col coordinates. The row coordinate changes in the vertical direction and the col coordinate in the horizontal direction, which means the x coordinate should associate with a column on the zone grid and the y coordinate is associated with a row on the zone grid, but this sometimes feels counter-intuitive. Hence, we needed to double check that we communicated the coordinates the correct way between the Web App and RPi.

For the forbidden zone task, we originally thought about sending a dictionary of coordinates as the keys and if the coordinate is forbidden or not as the values. But, we realized there was extra data being sent, and that only the forbidden zone coordinates need to be sent from the Web App to the RPi. Similarly for the activity logs tasks, we originally had in mind to send a dictionary of coordinates as the keys and the amount of frames spent on each coordinate as the values. But, this led to extra data being sent from the RPi to the Web App as we only need to send the amount frames spent on each coordinate of the room picture since it was already predetermined that the order of the data would be from the top left to the bottom right of the room picture. To create a successful overall system, testing done on the Web App using a test double as the CV was important as if the Web App could pass these tests, then integrating with the real CV and RPi was much simpler. As well, in person integration of the Web App and CV was important in creating the overall system of the product.

7 TEST & VALIDATION

Note: all results listed are from tests conducted using laptops as the primary processing device, and not the RPi. Full RPi tests are still in progress, although preliminary observation shows that it is a bit slower/has lower frame rate, as expected.

Test results are listed as: Test, Method, Metric, Goal, Result.

7.1 Results for Zone Detection Speed Design Specification

Zone notification speed	Slow-Mo Video	Pet enters zone in real life -> Web app notification	< 10 seconds	~1.125 seconds

We wanted to ensure users are able to receive notifications quickly if pets enter a forbidden zone as users do not wants pets inside those designated forbidden zones. The first step to this is testing the pet actually entering the forbidden zone in real life to the CV detecting this action occurring. The method used for this is a slow motion camera as it can precisely time from the instant the pet enters the forbidden zone in real life to the instant the CV detects this occurring. Our design requirement as specified above was for this to occur in less than one second. The results showed that this occurred on average in 0.625 seconds, which meant we reached our goal. We determined that the limiting factor is frame rate of the CV. Even though this design specification does not include requests to the Web App, the CV will always being sending requests while detecting video frame changes. As these requests are not instantaneous, this slows the frame rate at which pixel changes are detected, which leads to a slow down of zone detection speed.

7.2 Results for Notification Speed Use-Case Specification

Zone notification speed	Slow-Mo Video	Pet enters zone in real life -> Web app notification	< 10 seconds	~1.125 seconds

The next logical step in testing notification speed is how fast data could be sent from the CV to the Web App. Hence, our test is from the pet entering the forbidden zone in real life to the user receiving the notification on the Web

App. This result will include the time taken from the pet entering the forbidden zone in real life to the CV detecting this action as specified above. The method is similar to the design requirement for notification speed as specified above, which is using a slow motion camera to track the time from pet entering the forbidden zone in real life to the user receiving the notification on the Web App. Though this use-case requirement included the ML component into the product, we wanted a less than 10 second notification speed for this test. The results gave us a 1.125 second notification speed on average. The biggest limiting factor for this test was the polling rate, or the rate at which requests from the frontend to the backend occur. In this test, we had a 1 request per second polling rate. If we had a 1 request per 3 second polling rate, then the notification speed would be around 3 seconds. Hence, the notification speed is highly dependent on the polling rate. But, even if a faster polling rate means faster notifications for the user, this would waste many server resources on requests when in general each request would ending up resulting in no notification popup.

7.3 Results for Tracking Accuracy Design Specification

Accuracy of tracking	Human vs. Computer	Distance between centers of human-chosen vs. computer-chosen bounding boxes	Within 1 foot	Generally 3-6 inches

An important component of tracking pet movement is the accuracy with which the CV algorithm can accomplish this. The method to testing this is comparing the CV generated bounding box to a human-chosen bbox. A sampling of frames (and the bboxes that the CV code had generated at that point) are saved, and then presented to the tester once the live video terminates. The tester selects where they think the region of interest is, then the difference in bbox centers is displayed the distance is estimated (based on known distances in the photo). On average, we wanted the distance to be within one foot. After testing, we found the distance to be 3-6 inches on average.

As with any CV task, the limiting factor on accuracy is largely the frame rate. Slower frame rate leads to larger jumps in position between frames, which is harder for the OpenCV trackers to follow and thus is more likely to cause them to fail. It also leads to bigger perceived motion, and thus less precise overall bboxes. One major factor which slows down the frame rate is how frequently we make requests to the web server. Particularly with live video, which must be sent every frame, this slows down how quickly we can process a frame. Parallelism may be one solution to this problem.

7.4 Results for New Animal Detection Speed Design Specification

Detection speed of new animals	Slow-Mo Video	Animal enters frame in real life -> bounding box appears	< 5 seconds	~0.75 seconds

For forbidden zones on the edges of the room, detection speed on pets entering the frame is important so forbidden zone notifications can be sent quickly to the user. Hence, we tested the detection speed of new animals entering the camera frame. Similar to the method for notification speed, we used a slow motion camera that tracked the time from the pet entering the frame in real life to the bbox appearing on the CV. As specified in the design requirements earlier, we wanted this detection speed to occur under 5 seconds. On average, we found from tests that the detection speed was around 0.75 seconds, which beat our goal significantly. Like the CV detection speed from the notification speed design test, the limiting factor once again is the frame rate of the CV. The CV will always being sending requests while detecting camera frame changes. These requests are not instantaneous, which means the frame rate at which pixel changes are detected will be slower, which leads to a slower detection speed.

7.5 Results for User Accessibility Use Case Specification

In terms of user costs, we wanted users to be able to purchase the product in under $100 as this would make our product competitive with other similar products. In the end, the total user cost ended up being $88 as the user would need a Raspberry Pi 4, RPi Camera, and SD Card. Another test we wanted to run is user testing, specifically for user experience on the Web App and system setup. Though we were not able to get the full 10 participants, we had preliminary results for user experience and system setup. Overall, it seemed that people were able to setup the camera and the product, and that people were able to navigate the Web App and do the main tasks on the Web App. But, there were some suggestions on improving the user experience in terms of navigating the Web App and more explaining on what to do for each task on the dashboard. Hence, we tentatively reached our goal for user testing, but there needs to be more work done in terms of testing on more participants and fixing the user experience of the Web App.

8 PROJECT MANAGEMENT

8.1 Schedule

The updated schedule from the design report to the final report is shown in Fig. 10. The major change is the removal of the ML component due to one member's personal setbacks. We did all the tasks that were mentioned

in the schedule, but we finished the tasks. especially Web App and integration, later than intended on the schedule.

8.2 Team Member Responsibilities

From the design report, the responsibilities of the members did not change significantly except for the removal of the ML component due to a member's personal setbacks and Brandon did not need to setup a server on the RPi as we decided not to run a server on the RPi. To reiterate, Rebecca was responsible for setting up the Raspberry Pi and working on the Computer Vision component, such as being able to detect movement from each pet in a room. Brandon was in charge of setting up the front end and back end of the web application. All members helped with integrating and testing the components.

8.3 Bill of Materials and Budget

Shown in Table 2 is our bill of materials and budget. We have labelled each cost as a project cost (which would come out of our $600 budget) and/or a user cost, which would contribute towards our hypothetical retail price. The RPi will be from the ECE500 inventory, and so has a project cost of $0. As shown, we anticipate that our overall use of budget will be $25, and the overall user-facing cost of our system will be roughly $88. From the design report, we did not end up buying a speaker as we believed this was not a major component of the product, so we would only work on implementing this after finishing the other components of the product. But, we ended up not having enough time, so we did not buy a speaker. We did not need to buy a domain name as the front end deployment was supplied with a given domain name. We added a SD Card to the bills of materials and budget as it is necessary for a user that uses a RPi. There was no added project cost due to a team member already owning a SD Card.

8.4 Risk Management

Earlier in the semester, the biggest risk was between using a RPi and Jetson. The Jetson would allow us to speed up the ML algorithm training needed for the product, but the RPi is significantly cheaper than the Jetson and would allow us to fulfill the user requirement of the product being under $100 for the user. But, the biggest risk shifted to how we would re-scope the product due to one member's personal setbacks. We considered all of the possible situations we could do to finish the product, such as trying to fully integrate all three major systems or integrating the Web App and CV on a RPi and having a stand alone ML component on the Jetson. The worst case scenario was to rescope the project by generalizing the tasks that could be done by a user to all the pets in the room. For example, a user was able to create forbidden zones for each pet in a room given our original design concept. In the worst case scenario, a user would only be able to create one forbidden zone for all the pets in a room. The team was in constant communication throughout the one member's personal setback so that we could re-assess what path we wanted to take for the product. In the end, we were forced to going through with the worst case scenario due to the member's personal setbacks. We removed from the project all the tasks with the Web App and CV that pertained to the ML, and we were able to create a product using the Web App and CV only that allowed a user to monitor all the pets in a room.

9 ETHICAL ISSUES

The main ethical issue is security issues related to private user data being stored and displayed on a public website. To start, a user can enter pet data such as a pet name and pet pictures into the Web App. More private data that needs to be given is a live video camera feed of the room and creating a room picture that will be stored and displayed for the other tasks on the Web App. Malicious users can watch these live video feeds if the correct security measures are not taken. This would be an ethical issue as private data should not be accessible by the general public. To mitigate this issue, we have an authentication system using JWT Authentication that requires users to have an access token to make requests to the frontend and/or backend of the Web App. As well, users need to know the serial number of the RPi as this number is used for storing and retrieving information from the database on the Web App.

10 RELATED WORK

There are currently many pet monitors on the market. One most similar to ours is this Nest Cam[4] made by Google. It has a cost of $99.99 and advertises a similar alert system, object identification, and data security. Specifically, it's identification can differentiate between a human, animal, or vehicle.

A cheaper existing option is the Wyze Cam v3 Pet Camera[1] which simply sends users notifications whenever motion or sound is detected and allows users to talk to animals through a microphone or play a sound through a speaker. The detection options on this one are much less than those of the Google option, but it is also only $35.98, which is almost a third of the price of the Google option.

11 SUMMARY

The intention of PetSTAR is to benefit pet owners with tasks that make taking care of their pets easier, while still including the current features that a traditional camera offers. The system contains two major components, specifically a Web App and CV algorithms, that interact together to support the purpose of helping pet owners keep tabs on their pets. We were able to meet the design specifications, omitting specifications that were based on the product with the ML component. Some limiting factors of the product

Table 2: Bill of Materials and Budget

Description	Model	Manufacturer	Project Cost	User Cost
Raspberry Pi 4	4GB	Raspberry Pi Foundantion	$0	$55
RPi Camera	3	Raspberry Pi Foundantion	$25	$25
SD Card	32GB	SanDisk	$0	$8
			$25.00	$88.00

include the rate at which we can request to and from the deployment server. This goes for communications between the RPi and the backend, as well as between the backend and frontend. The deployment server for the backend can only handle so many requests, which is why for example the live video feed looks smooth locally but is slower when using the deployment server. In addition, increasing the number of requests made by the CV code leads to a slower frame rate. To increase performance, one possibility for future work would be to include asynchronous requests and/or threading in the CV code so that the CV can parse the data from the camera while simultaneously making or getting requests from the database. Another suggestion is using an API that can display live video feed, which may increase the frame rate of the video feed on the Web App.

11.1 Lessons Learned

One major lesson our team learned was dealing with major setbacks and re-scoping the product such that it would fit our strengths but also capture the original design idea that we had at the beginning. It is important to have good and constant team communication so everyone know what everyone else is capable of and what readjustments we need to make based on the current situation. Other lessons include learning how to design and implement multiple subsystems of a product where each person is working on their own system. Each individual knows their sub-system well, but explaining the thought process behind each system to another team member that does not know the system well can be hard to do, which is why integration can be difficult and why there needs to be a large amount time dedicated to integration.

Glossary of Acronyms

- bbox - Bounding box
- CV - Computer Vision
- ML – Machine Learning
- RPi – Raspberry Pi
- Web App - Web Application

References

[1] Chewy. "WYZE Cam V3 Pet Camera - Chewy.com." Chewy.com, 15 Apr. 2023, tinyurl.com/49syhur9.

[2] Chitlangya, Ronak. "JWT Authentication with React JS and Django." Medium, Medium, 24 Apr. 2023, https://medium.com/@ronakchitlangya1997/jwt-authentication-with-react-js-and-django-c034aae1e60d.

[3] "Getting Started." Getting Started, Axios Docs, https://axios-http.com/docs/intro.

[4] Google Store. "Nest Cam (Indoor, Wired)." Google Store, tinyurl.com/5f9msp2u.

[5] Grinberg, Miguel. "Video Streaming with Flask." Miguelgrinberg.com, https://blog.miguelgrinberg.com/post/video-streaming-with-flask.

[6] Huls, Mike. "Detecting Motion With OpenCV — Image Analysis for Beginners." Medium, 16 Aug. 2022, towardsdatascience.com/image-analysis-for-beginners-creating-a-motion-detector-with-opencv-4ca6faba4b42.

[7] Irabor, Jordan. "How to Build A to-Do Application Using Django and React." DigitalOcean, DigitalOcean, 17 Feb. 2021, https://www.digitalocean.com/community/tutorials/build-a-to-do-application-using-django-and-react.

[8] Mallick, Satya. "Object Tracking Using Opencv (c++/Python)." LearnOpenCV, 11 Nov. 2022, https://learnopencv.com/object-tracking-using-opencv-cpp-python/.

[9] "OpenCV Modules." OpenCV, https://docs.opencv.org/4.x/index.html.

[10] Rosebrock, Adrian. "Basic Motion Detection and Tracking With Python and OpenCV - PyImageSearch." PyImageSearch, 11 Apr. 2023, pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv.

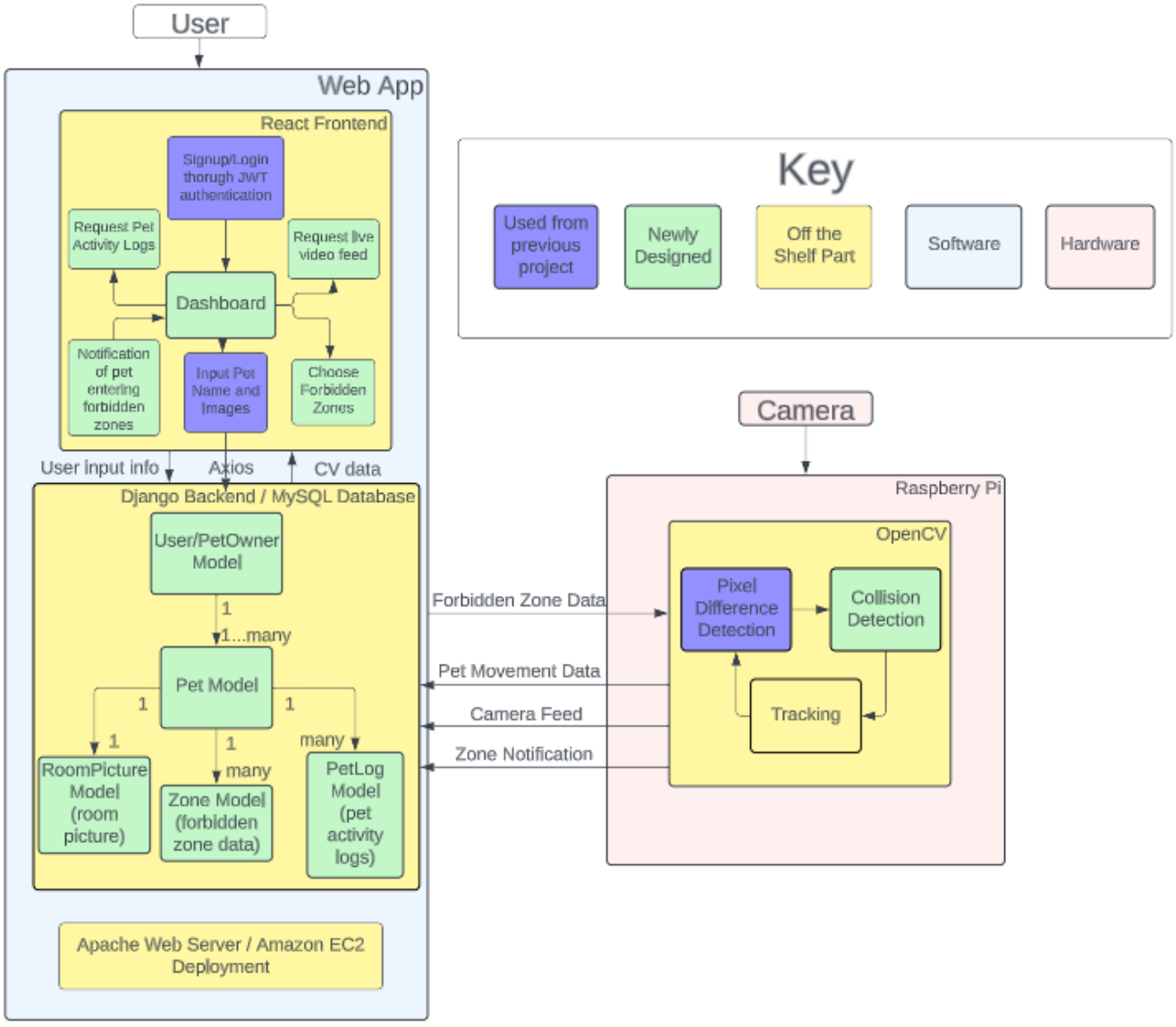

Figure 9: Full block diagram demonstrating where parts came from

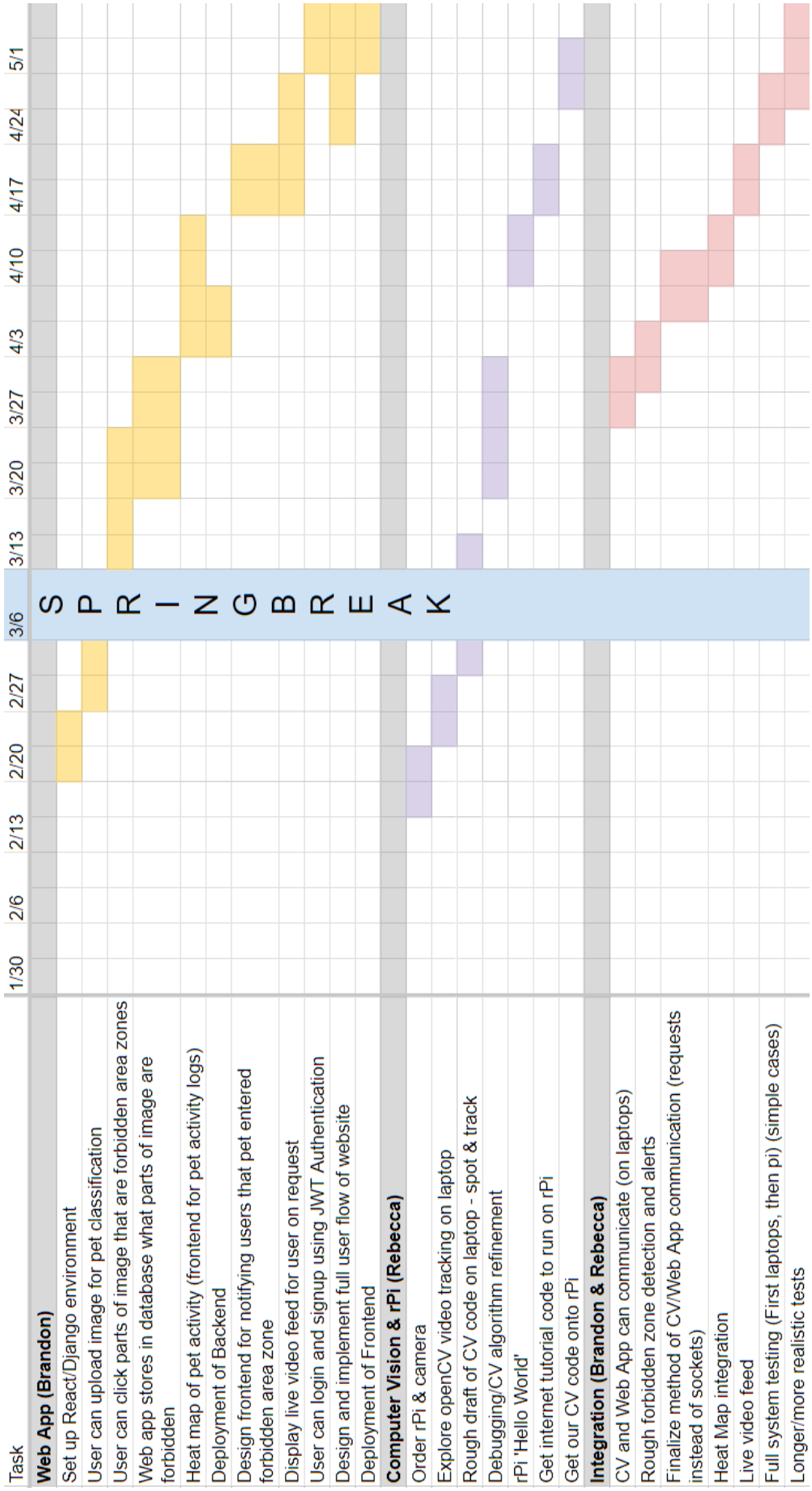

Figure 10: Gantt Chart