

# The vet's surgery

Sara Humphries

- Brief
- Initial & extension planning
- Demo
- Highlights: looping through a list to build a dictionary.

# The vet's surgery - brief

## Vet Management App

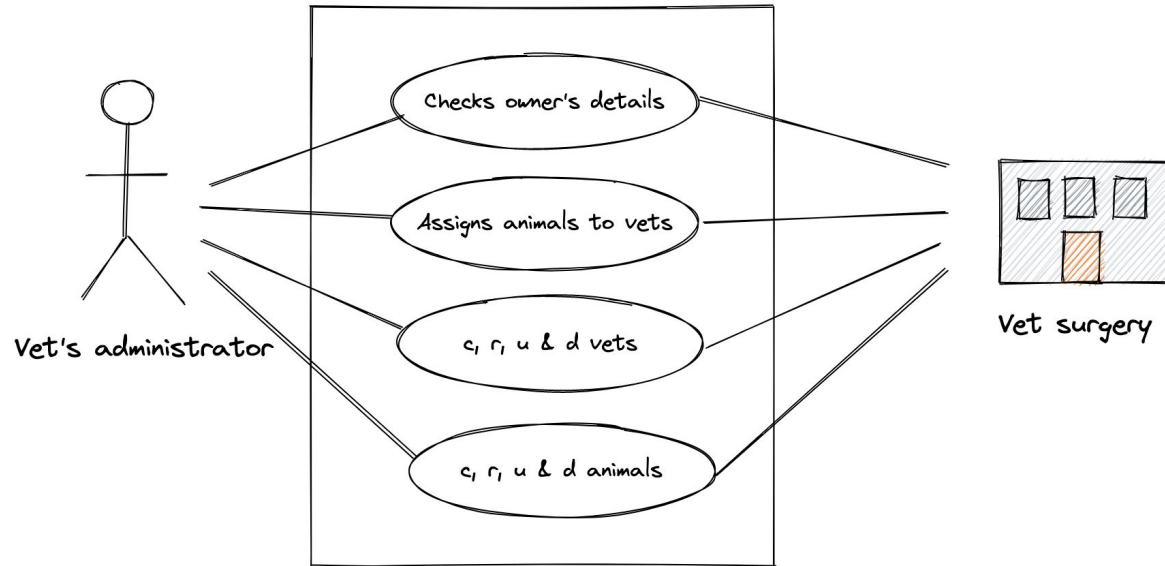
A veterinary practice has approached you to build a web application to help them manage their animals and vets. A vet may look after many animals at a time. An animal is registered with only one vet.

### MVP

- The practice wants to be able to register / track animals. Important information for the vets to know is -
  - Name
  - Date Of Birth (use a VARCHAR initially)
  - Type of animal
  - Contact details for the owner
  - Treatment notes
- Be able to assign animals to vets
- CRUD actions for vets / animals - remember the user - what would they want to see on each View? What Views should there be?

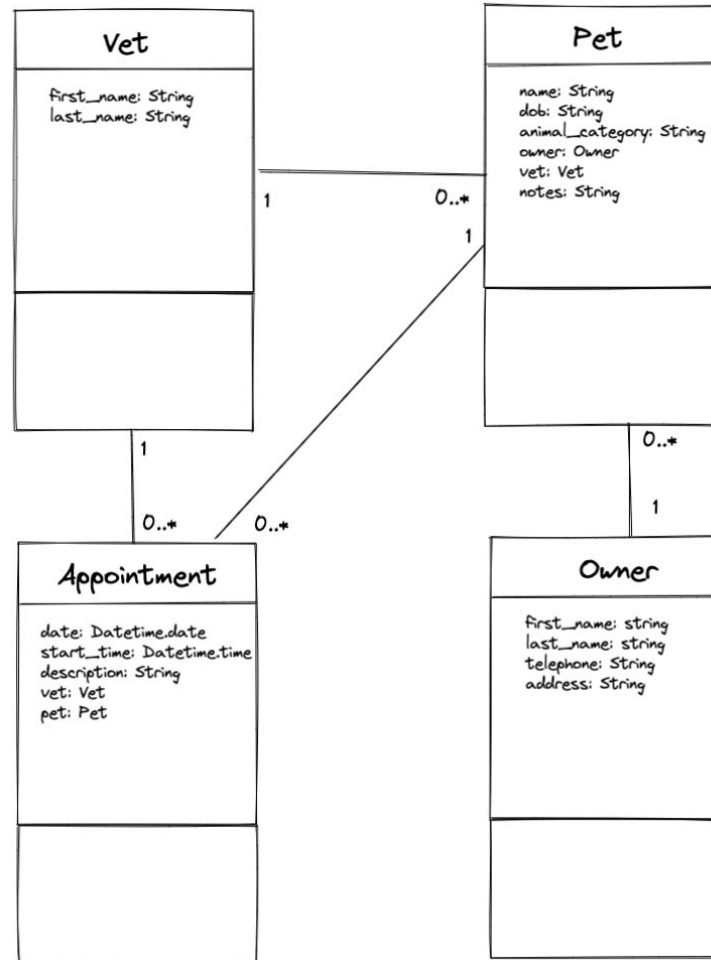
# The vet's surgery - MVP planning

## Case Diagram - Vets (MVP)



# Class diagrams

## Database tables



# Wireframes - Vets (MVP)

homepage

Home	Vet surgery name	nav bar
		nav bar
<div>Vets</div> <div>Pets</div>		
contact us telephone		

vets index / create / delete

Vets	Vet surgery name	nav bar
		nav bar
<div>Add new vet</div> <div>ANNA <span>✗ Delete Anna</span></div> <div>see details</div> <div>BILL <span>✗ Delete Bill</span></div> <div>see details</div>		
contact us telephone		

pets index / create / delete

Pets	Vet surgery name	nav bar
		nav bar
<div>Add new pet</div> <div>Woof: Dog <span>✗ Delete Woof</span></div> <div>Owner: R. Truman</div> <div>Vet: Anna</div> <div>see details</div> <div>Sevens: Small animal <span>✗ Delete Sevens</span></div> <div>Owner: T. Smith</div> <div>Vet: NONE</div> <div>see details</div>		
contact us telephone		

vets show

Anna	Vet surgery name	nav bar
		nav bar
<div>First name: Anna</div> <div>Last name: Hall</div> <div>Pets:</div> <div>- Woof</div> <div>- Charlie</div> <div>edit Anna's details</div>		
contact us telephone		

pets show

Woof	Vet surgery name	nav bar
		nav bar

pets edit/update

Woof	Vet surgery name	nav bar
		nav bar
<div>Name: (dropdown)</div> <div>Category: (dropdown)</div> <div>DOB: (date picker)</div> <div>Owner: (dropdown)</div> <div>Vet: (dropdown)</div> <div>Notes:</div> <div>is a lovely dog</div> <div>had a broken leg once</div> <div>save woof's details</div>		
contact us telephone		

owners show

R. Truman	Vet surgery name	nav bar
		nav bar
<div>First name: Rob</div> <div>Last name: Truman</div> <div>telephone: 0115 841 2772</div> <div>address:</div> <div>21, Millar Crescent</div> <div>B40 5HW</div> <div>Pets: Woof</div>		
contact us telephone		

vets edit / update

Anna	Vet surgery name	nav bar
		nav bar
<div>First name: (text)</div> <div>Last name: (text)</div> <div>Pets:</div> <div>- Woof</div> <div>- Charlie</div> <div>save Anna's details</div>		
contact us telephone		

# Extension planning Wireflow

## Demo



# Fav code = looping through a list to aggregate data more usefully

**Problem:** Rather than see the appointments all listed underneath each other, I want to group them by date to look more like a diary

[Home](#)  
[Vet's database](#)  
[Pet's database](#)  
[Customer database](#)

The Vet's Surgery

All appointments

08/06/2021

9am: Fido with R. Truman  
9am: Posy with D. Smith  
11am: Squeak with D. Smith

21/07/2021

9am: Fido with R. Truman  
9am: Posy with D. Smith  
11am: Bunny with D. Smith

Book another appointment

Back to main page

Go to page 2

The Vet's Surgery

# Fav code = looping through a list to aggregate data more usefully

Example: I collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)



# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first colour. Create a key 'red' with a count of 1

`{ 'red': 1 }`

# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first item. Create a key 'red' with a count of 1
2. Take the next item. Check whether 'blue' exists as a key in the dictionary. No, so create a key 'blue' with count 1

`{ 'red': 1 }`

# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first item. Create a key 'red' with a count of 1
2. Take the next item. Check whether 'blue' exists as a key in the dictionary. No, so create a key 'blue' with count 1

```
{ 'red': 1,  
  'blue': 1 }
```

# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first item. Create a key 'red' with a count of 1
2. Take the next item. Check whether 'blue' exists as a key in the dictionary. No, so create a key 'blue' with count 1
3. Take the next item 'blue'. Check whether 'blue' exists as a key in the dictionary. Yes! Increase the count to 2

```
{ 'red': 1,  
  'blue': 1 }
```

# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first item. Create a key 'red' with a count of 1
2. Take the next item. Check whether 'blue' exists as a key in the dictionary. No, so create a key 'blue' with count 1
3. Take the next item 'blue'. Check whether 'blue' exists as a key in the dictionary. Yes! Increase the count to 2

```
{ 'red': 1,  
  'blue': 2 }
```

# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first item. Create a key 'red' with a count of 1
2. Take the next item. Check whether 'blue' exists as a key in the dictionary. No, so create a key 'blue' with count 1
3. Take the next item 'blue'. Check whether 'blue' exists as a key in the dictionary. Yes! Increase the count to 2
4. Take the.....

```
{ 'red': 1,  
  'blue': 2 }
```

# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first item. Create a key 'red' with a count of 1
2. Take the next item. Check whether 'blue' exists as a key in the dictionary. No, so create a key 'blue' with count 1
3. Take the next item 'blue'. Check whether 'blue' exists as a key in the dictionary. Yes! Increase the count to 2
4. Take the.....

```
{ 'red': 1,  
  'blue': 2,  
  'grey': 1},
```

# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first item. Create a key 'red' with a count of 1
2. Take the next item. Check whether 'blue' exists as a key in the dictionary. No, so create a key 'blue' with count 1
3. Take the next item 'blue'. Check whether 'blue' exists as a key in the dictionary. Yes! Increase the count to 2
4. Take the.....

```
{ 'red': 2,  
  'blue': 2,  
  'grey': 1},
```



# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first item. Create a key 'red' with a count of 1
2. Take the next item. Check whether 'blue' exists as a key in the dictionary. No, so create a key 'blue' with count 1
3. Take the next item 'blue'. Check whether 'blue' exists as a key in the dictionary. Yes! Increase the count to 2
4. Take the.....

```
{ 'red': 2,  
  'blue': 2,  
  'grey': 1,  
  'yellow': 1}
```

# Fav code = looping through a list to aggregate data more usefully

Example: If collected some data on the colour of cars that passed outside the CodeBase building yesterday:

`['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']`

I want to find the mode (most common) colour (assuming I don't know all the colours that a car might be)

1. Take the first item. Create a key 'red' with a count of 1
2. Take the next item. Check whether 'blue' exists as a key in the dictionary. No, so create a key 'blue' with count 1
3. Take the next item 'blue'. Check whether 'blue' exists as a key in the dictionary. Yes! Increase the count to 2
4. Take the.....

```
{ 'red': 2,  
  'blue': 3,  
  'grey': 1,  
  'yellow': 1}
```

```
1
2
3 car_colours = ['red', 'blue', 'blue', 'grey', 'red', 'yellow', 'blue']
4
5 agg_data = {}
6 for colour in car_colours:
7     if colour in agg_data.keys():
8         agg_data[colour] += 1
9     else:
10        agg_data[colour] = 1
11
12 print(agg_data)
13
14
15
16
17
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

1: zsh

```
→ planning git:(main) x python3 presentation.py
{'red': 2, 'blue': 3, 'grey': 1, 'yellow': 1}
→ planning git:(main) x █
```

```
def appointments_by_day(self):
    ordered_appointments = sorted(self.appointments, key=lambda k: k.start_time)
    diary = []
    for appointment in ordered_appointments:
        placed = False
        for day in diary:
            if appointment.date == day['date']:
                day['appointments'].append(appointment)
                placed = True
        if not placed:
            day = {
                'date': appointment.date,
                'appointments': [appointment]
            }
            diary.append(day)

    ordered_diary = sorted(diary, key=lambda k: k['date'])
    return ordered_diary
```

Loop through  
appointments

If appointment date is in  
there already, append  
to list

If not, make a whole  
new date

```
def appointments_by_day(self):
    ordered_appointments = sorted(self.appointments, key=lambda k: k.start_time)
    diary = []
    for appointment in ordered_appointments:
        placed = False
        for day in diary:
            if appointment.date == day['date']:
                day['appointments'].append(appointment)
                placed = True
        if not placed:
            day = {
                'date': appointment.date,
                'appointments': [appointment]
            }
            diary.append(day)

    ordered_diary = sorted(diary, key=lambda k: k['date'])
    return ordered_diary
```

Loop through  
appointments

If appointment date is in  
there already, append  
to list

If not, make a whole  
new date

```
[ { 'date': '2021-06-10', 'appointments': appointment_1 } ]
```

```
def appointments_by_day(self):
    ordered_appointments = sorted(self.appointments, key=lambda k: k.start_time)
    diary = []
    for appointment in ordered_appointments:
        placed = False
        for day in diary:
            if appointment.date == day['date']:
                day['appointments'].append(appointment)
                placed = True
        if not placed:
            day = {
                'date': appointment.date,
                'appointments': [appointment]
            }
            diary.append(day)

    ordered_diary = sorted(diary, key=lambda k: k['date'])
    return ordered_diary
```

Loop through  
appointments

If appointment date is in  
there already, append  
to list

If not, make a whole  
new date

```
[ { 'date': '2021-06-10', 'appointments': appointment_1 },
  { 'date': '2021-06-11', 'appointments': appointment_2 }
]
```

```
def appointments_by_day(self):
    ordered_appointments = sorted(self.appointments, key=lambda k: k.start_time)
    diary = []
    for appointment in ordered_appointments:
        placed = False
        for day in diary:
            if appointment.date == day['date']:
                day['appointments'].append(appointment)
                placed = True
        if not placed:
            day = {
                'date': appointment.date,
                'appointments': [appointment]
            }
            diary.append(day)

    ordered_diary = sorted(diary, key=lambda k: k['date'])
    return ordered_diary
```

Loop through  
appointments

If appointment date is in  
there already, append  
to list

If not, make a whole  
new date

```
[ { 'date': '2021-06-10', 'appointments': appointment_1, appointment_3 },
  { 'date': '2021-06-11', 'appointments': appointment_2 }
]
```

```
def appointments_by_day(self):
    ordered_appointments = sorted(self.appointments, key=lambda k: k.start_time)
    diary = []
    for appointment in ordered_appointments:
        placed = False
        for day in diary:
            if appointment.date == day['date']:
                day['appointments'].append(appointment)
                placed = True
        if not placed:
            day = {
                'date': appointment.date,
                'appointments': [appointment]
            }
            diary.append(day)

    ordered_diary = sorted(diary, key=lambda k: k['date'])
    return ordered_diary
```

Loop through  
appointments

If appointment date is in  
there already, append  
to list

If not, make a whole  
new date

```
[ { 'date': '2021-06-10', 'appointments': appointment_1, appointment_3 },
  { 'date': '2021-06-11', 'appointments': appointment_2 },
  { 'date': '2021-06-15', 'appointments': appointment_4 }
]
```



```

def appointments_by_day(self):
    ordered_appointments = sorted(self.appointments, key=lambda k: k.start_time)
    diary = []
    for appointment in ordered_appointments:
        placed = False
        for day in diary:
            if appointment.date == day['date']:
                day['appointments'].append(appointment)
                placed = True
        if not placed:
            day = {
                'date': appointment.date,
                'appointments': [appointment]
            }
            diary.append(day)

    ordered_diary = sorted(diary, key=lambda k: k['date'])
    return ordered_diary

```

Loop through appointments

If appointment date is in there already, append to list

If not, make a whole new date

### Appointments schedule

2021-06-10

Time: 2	Pet: Francois	Vet: Bill, Osman	<a href="#">Edit appointment time/date</a>	<button>Cancel appointment</button>
---------	---------------	------------------	--	-------------------------------------

Time: 4	Pet: Harrison	Vet: Kirsty, White	<a href="#">Edit appointment time/date</a>	<button>Cancel appointment</button>
---------	---------------	--------------------	--	-------------------------------------

2021-06-11

Time: 6	Pet: Red	Vet: Carl, Driver	<a href="#">Edit appointment time/date</a>	<button>Cancel appointment</button>
---------	----------	-------------------	--	-------------------------------------

2021-06-15

Time: 6	Pet: Billy	Vet: Carl, Driver	<a href="#">Edit appointment time/date</a>	<button>Cancel appointment</button>
---------	------------	-------------------	--	-------------------------------------

*The End*





# The vet's surgery - Fav error

```
@pets_blueprint.route('/pets/new', methods = ['POST'])
def create_pet():
    name = request.form['name']
    dob = request.form['dob']
    animal_category = request.form['animal_category']
    owner_id = request.form['owner_id']
    vet_id = request.form['vet_id']
    notes = ""
```

**AttributeError: module 'werkzeug.wrappers.request' has no attribute 'form'**

# Fav error - top 3 google results

`AttributeError: module 'werkzeug.wrappers.request' has no attribute 'form'`

1 You confused the `requests` library with the `request` object from Flask. – Klaus D. Jul 8 '19 at 0:12



## Request / Response Objects

The request and response objects wrap the WSGI environment or the return value from a WSGI application so that it is another WSGI application (wraps a whole application).

### How they Work

Your WSGI application is always passed two arguments. The WSGI “environment” and the WSGI `start_response` function that is used to start the response phase. The **Request** class wraps the `environ` for easier access to request variables (form data, request headers etc.).

The **Response** on the other hand is a standard WSGI application that you can create. The simple hello world in Werkzeug looks like this:

```
from werkzeug.wrappers import Response
application = Response('Hello World!')
```

### Contents

[Request / Response Objects](#)

- [How they Work](#)
- [Mutability and Reusability of Wrappers](#)
- [Wrapper Classes](#)

### Navigation

[Overview](#)

[To make it more useful, you can decorate it with a function and do some processing.](#)

[readthedocs.org](https://readthedocs.org) › [werkzeug](#) › [downloads](#) › [pdf](#) [PDF](#)

## Werkzeug Documentation (0.15.x) - Read the Docs

26 Jan 2020 — If you are using Python 2, the `venv` module is not available. ... from `werkzeug.wrappers` import `Request`, `Response` ... `url = request.form['url']` ... Also common headers are exposed as `attributes` or with methods to set / retrieve ...