Evidence Gathering Document for SQA Level 8 Professional Developer Award.

This document is designed for you to present your screenshots and diagrams relevant to the PDA and to also give a short description of what you are showing to clarify understanding for the assessor.

Fill in each point with screenshot or diagram and description of what you are showing.

Each point requires details that cover each element of the Assessment Criteria, along with a brief description of the kind of things you should be showing.

Week 1

Unit	Ref	Evidence
I&T	I.T.6	Demonstrate the use of a hash in a program. Take screenshots of: *A hash in a program *A function that uses the hash *The result of the function running

Paste Screenshot here

```
def setup
  @drink_1 = Drink.new("Tennenets", 3.20, 4.2)
  @drink_2 = Drink.new("Smirnoff", 2.10, 15.0)
  @customer = Customer.new("Shaun", 95.50, 26)
  @customer2 = Customer.new("Craig", 33.10, 16)
  @stock_GG = {@drink_1 => 10, @drink_2 => 20}
  @pub = Pub.new("The Goughs Gallon", 250.55, @stock_GG)
end
```

```
  def stock_value
    total = 0
    for drink in @stock.keys
      total += (drink.price * @stock[drink])
    end
    return total
  end
```

```
Run options: --seed 23409

# Running:

........"The total stock value is 74.0"
```

Description here

Above (1) is a hash “stock_GG” being defined in Ruby. It has two keys of different drink types and relative values of stock.

The function (2) uses a for loop to take the values of each key and multiplies them by the third respective value in the instance variable of each Drink instance. It performs the calculations 4.2 x 10 in the first loop and adds this to the variable total. On the second loop it adds on to this 15.0 x 20. It then returns the total, 74.0.

The result (3) shows this function being called in an interpolated print method that returns this total.

Week 2

Unit	Ref	Evidence
I&T	I.T.5	Demonstrate the use of an array in a program. Take screenshots of: *An array in a program *A function that uses the array *The result of the function running

Paste Screenshot here

```ruby
def setup
  @fish_1 = Fish.new("Trout")
  @fish_2 = Fish.new("Pike")
  @fish = [@fish_1, @fish_2]
  @river_1 = River.new("Amazon", @fish)
end
```

```ruby
def fish_count
  return "There are #{fish.size} in the river."
end
```

```
[➜ specs git:(master) ✗ ruby river_spec.rb
"There are 2 in the river."
```

Description here

In (1) There is an array of Fish being stored in a River.

The function (2) returns a statement declaring the amount of Fish in the array in River.

The function is called on our river instance returning the statement (3).

Week 3

Unit	Ref	Evidence
I&T	I.T.3	Demonstrate searching data in a program. Take screenshots of: *Function that searches data *The result of the function running

Paste Screenshot here

```ruby
def self.all()
  sql = "SELECT * FROM customers;"
  customers = SqlRunner.run(sql)
  result = customers.map{|customer| Customer.new(customer)}
  return result
end
```

```
[2] pry(main)> Customer.all()
=> [#<Customer:0x007fdede02f340 @funds=42.52, @id=58, @name="Sara">,
 #<Customer:0x007fdede02f1d8 @funds=22.1, @id=57, @name="Johnny">]
```

Description here

The function (1) shows a ruby function that first takes an sql command as a string requesting sql to get all entries on the table “customers.” A customers variable is then assigned to the returned sql command placed in a sanatized runner. A result variable is then assigned as an array which converts each returned entry of a “customer” from sql table entries into a ruby hash. This array of hashes is then returned.

Our result (2) shows the result of this Function being used and returning all 2 customers and each column of data stored in sql assigned to a respective key-value hash pair.

Unit	Ref	Evidence
I&T	I.T.4	Demonstrate sorting data in a program. Take screenshots of: *Function that sorts data *The result of the function running

Paste Screenshot here

```
comics_project=# select * from publishers order by name;
```

```
 id | image |                      logo                      |     name
----+-------+------------------------------------------------+--------------
  7 |       | /images/Images/Publisher/BoomLogo.jpg          | Boom
  2 |       | /images/Images/Publisher/DCLogo.jpg            | DC
  4 |       | /images/Images/Publisher/DarkHorseLogo.jpg     | Darkhorse
 10 |       | /images/Images/Publisher/FantagraphicsLogo.jpg | Fantagraphics
  6 |       | /images/Images/Publisher/HumanoidsLogo.jpg     | Humanoids
  8 |       | /images/Images/Publisher/IDWLogo.jpg           | IDW
  3 |       | /images/Images/Publisher/ImageLogo.jpg         | Image
  1 |       | /images/Images/Publisher/Marvel.jpg            | Marvel
  9 |       | /images/Images/Publisher/ValiantLogo.jpg       | Valiant
  5 |       | /images/Images/Publisher/VertigoLogo.jpg       | Vertigo
(10 rows)
```

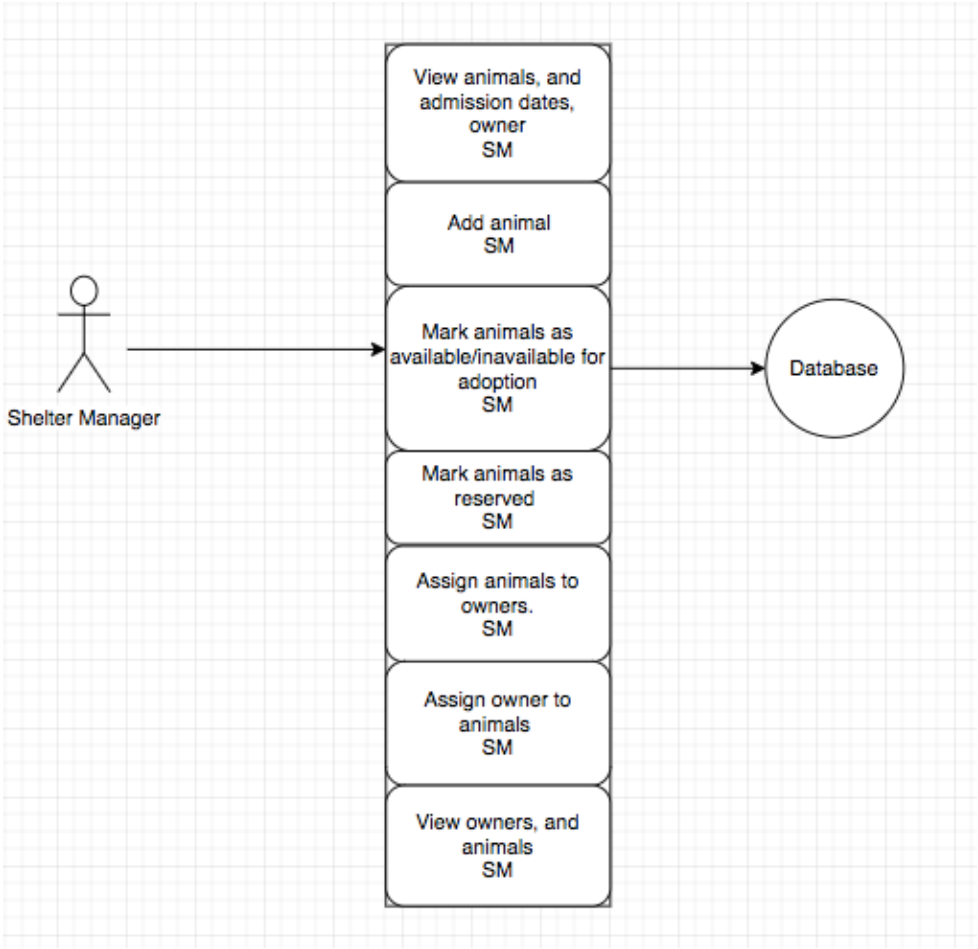
Description here

Above shows a sql function that calls for publishers in the database comics_project and orders them by name (in alphabetical order). The second picture shows the result of this function being run.

Week 4

Unit	Ref	Evidence
A&D	A.D.1	A Use Case Diagram

Paste Screenshot here

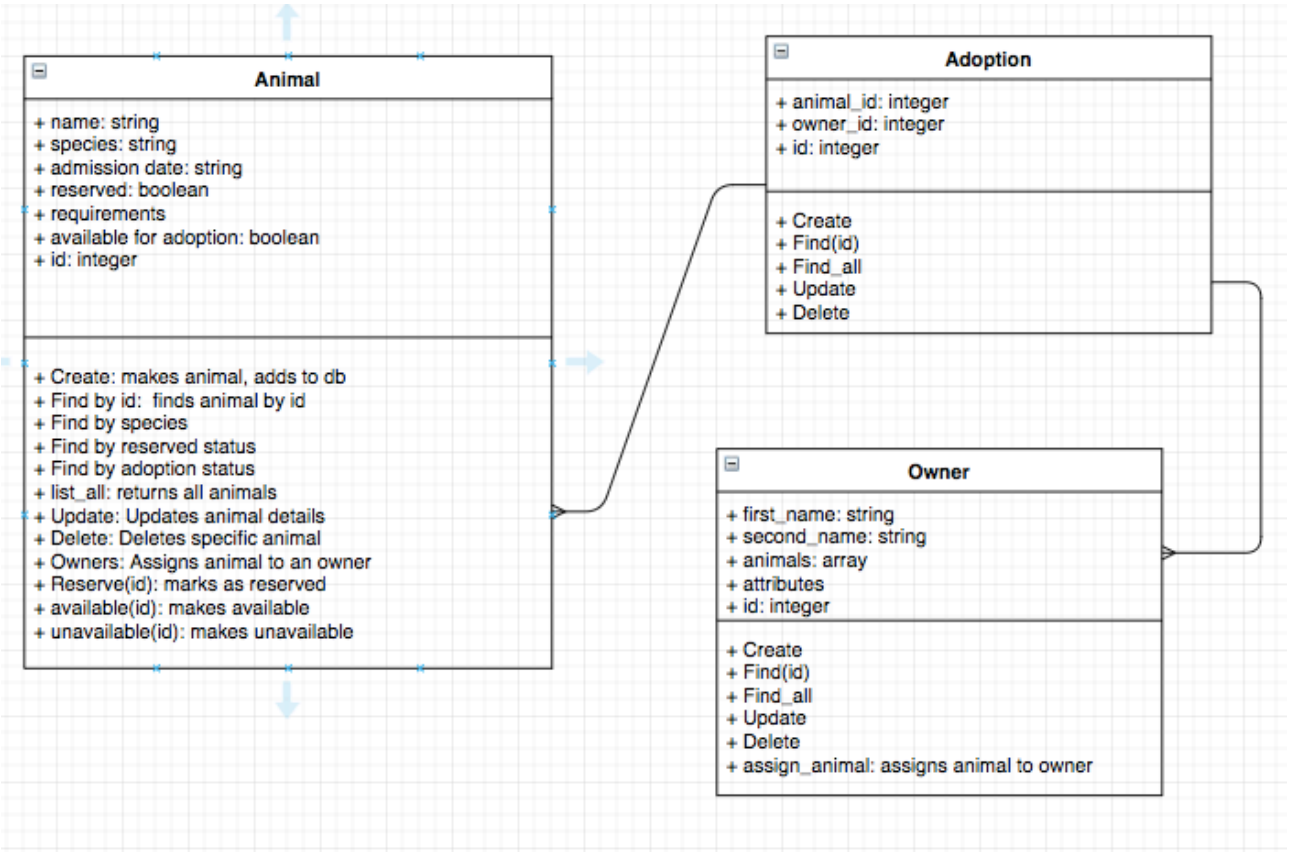


Description here

This Case Diagram shows a user “Shelter Manager and each function he should be able to use with the prospective software application. Each function then interacts with a central database which stores and/or returns information to the app.

Unit	Ref	Evidence
A&D	A.D.2	A Class Diagram

Paste Screenshot here



Description here

This Class Diagram shows three classes interacting in our prospective app. The animal and the Owner Class both interact with the Adoption class on a many to one basis respectively. Each Class representation in our Diagram has three sections. These show, from the top down:

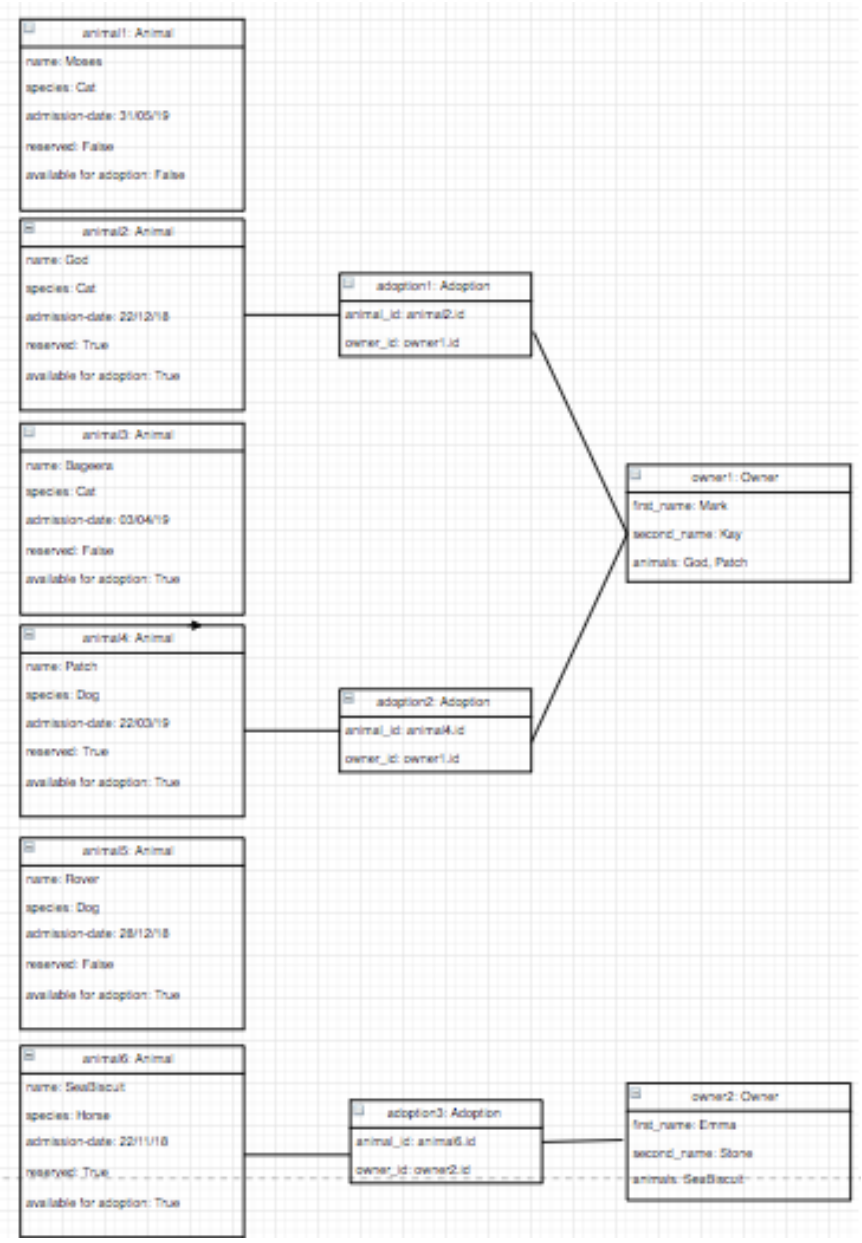
1) The Name of the Class

2) The variables that must be defined in each class instance and their respective data type

3) The methods available to each Class.

Unit	Ref	Evidence
A&D	A.D.3	An Object Diagram

Paste Screenshot here

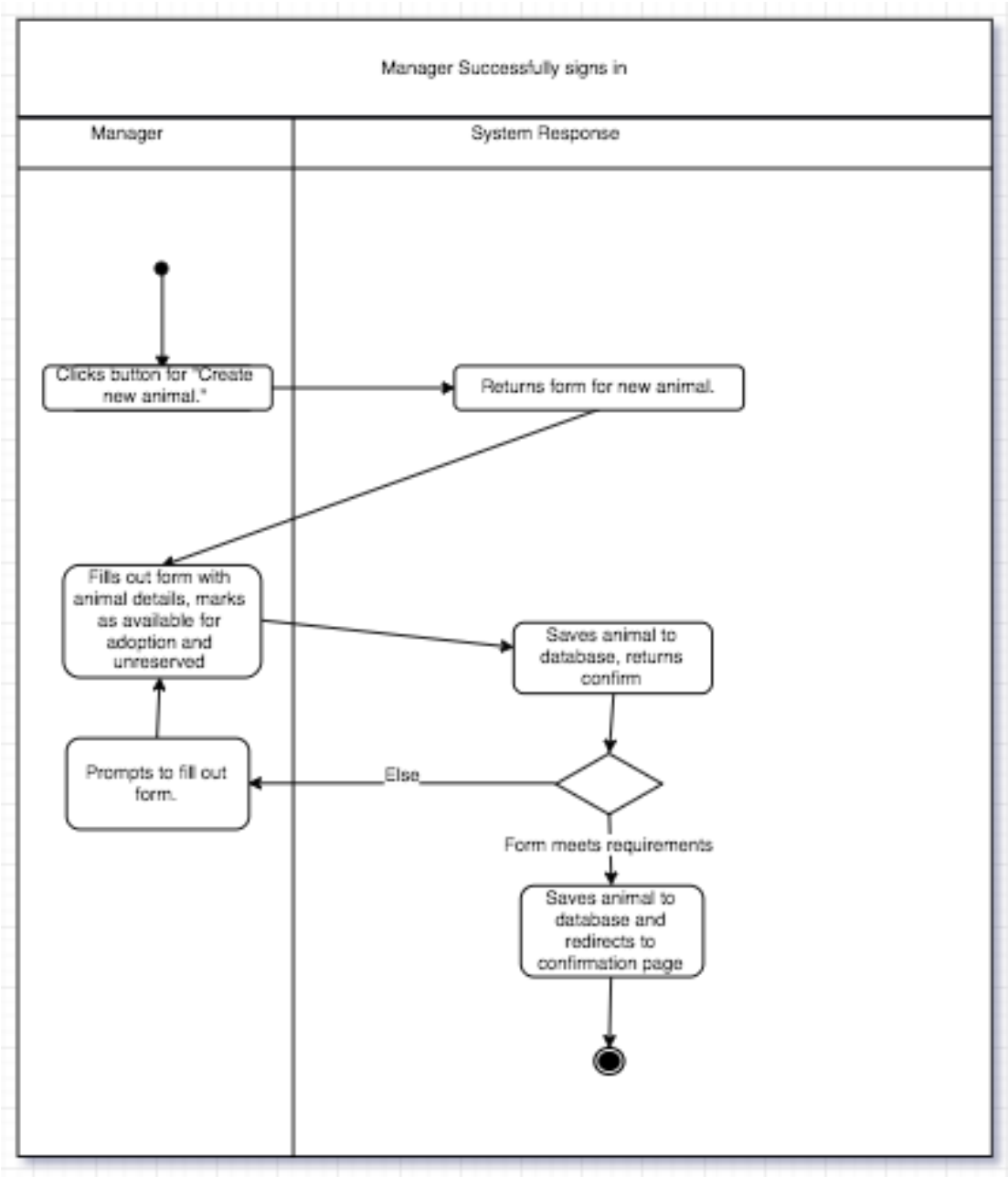


Description here

This Object Diagram shows specific instances of the Classes outlined in our Class diagram and which other instances of Classes they may relate to.

Unit	Ref	Evidence
A&D	A.D.4	An Activity Diagram

Paste Screenshot here



Description here

This Diagram shows one potential User Journey with our application. It details their own actions and responses and the system responses to this.

Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time

Paste Screenshot here

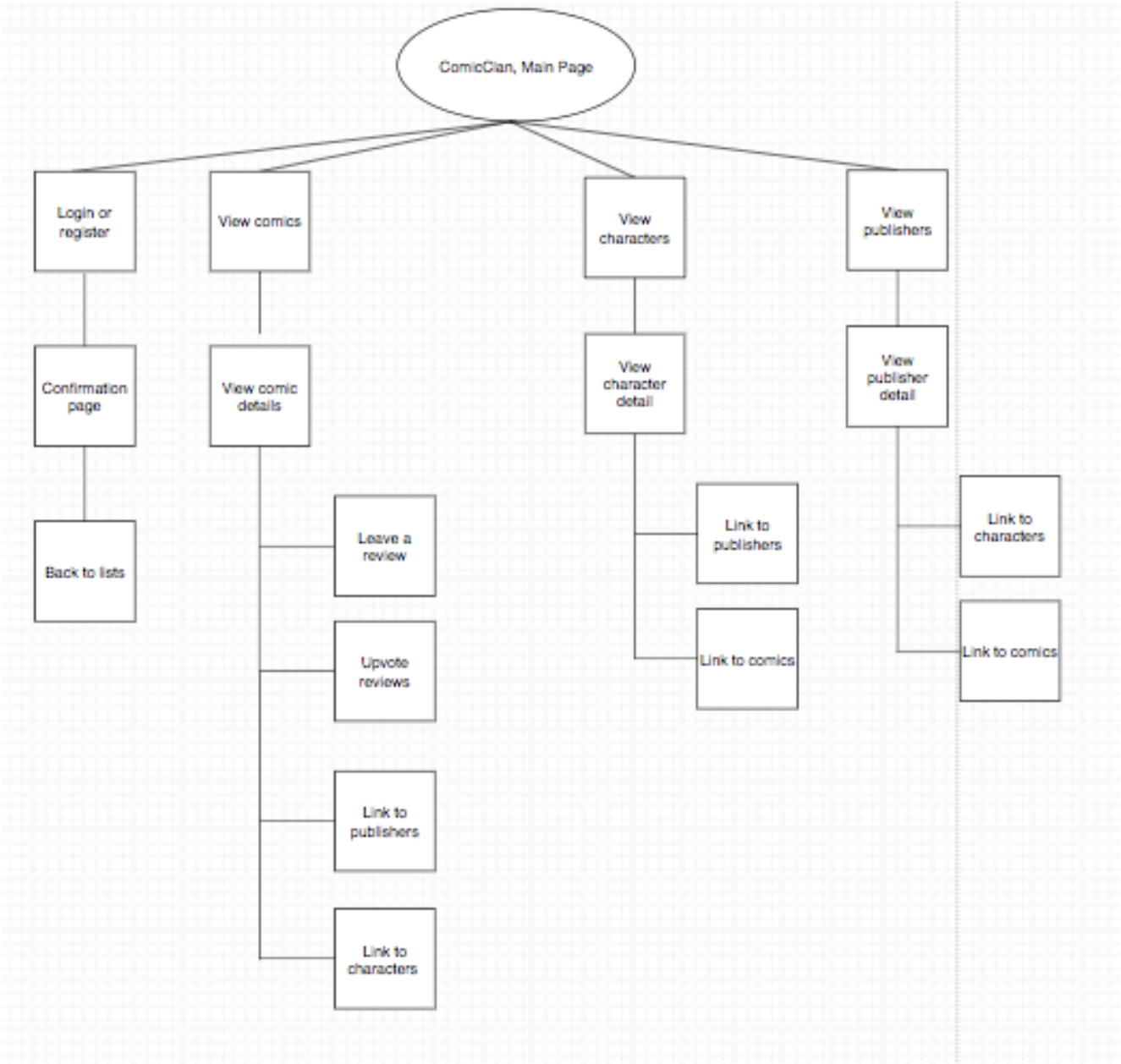
Topic	Possible effect and constraint on Product	Possible solution
Hardware and software constraints	Limited ram and processing power	Shouldn't affect simple app protoype
Performance requirements	Low performance requirements, simple web app	Shouldn't be an issue
Server problems	Potentially unlimited traffic	A select product for animal shelter managers, however if expanded to include potential owners, this could be an issue. Expand database, monitor growth and react
Usability	Visually impaired users, not technically proficient users	Include alt tags in html and use reasonably sized images. Have big tags for navigating site, use reasonable font sizes.
Budgets	No money	Shouldn't be an issue developing first prototype
Time limitations	Deadline 05/06/2019	Effective time management and allow time for debugging, UX adjustments and error.

Description here

This Diagram shows possible topics of constraint, what that would mean for our particular application and potential solutions to these constraints

Unit	Ref	Evidence
P	P.5	User Site Map

Paste Screenshot here

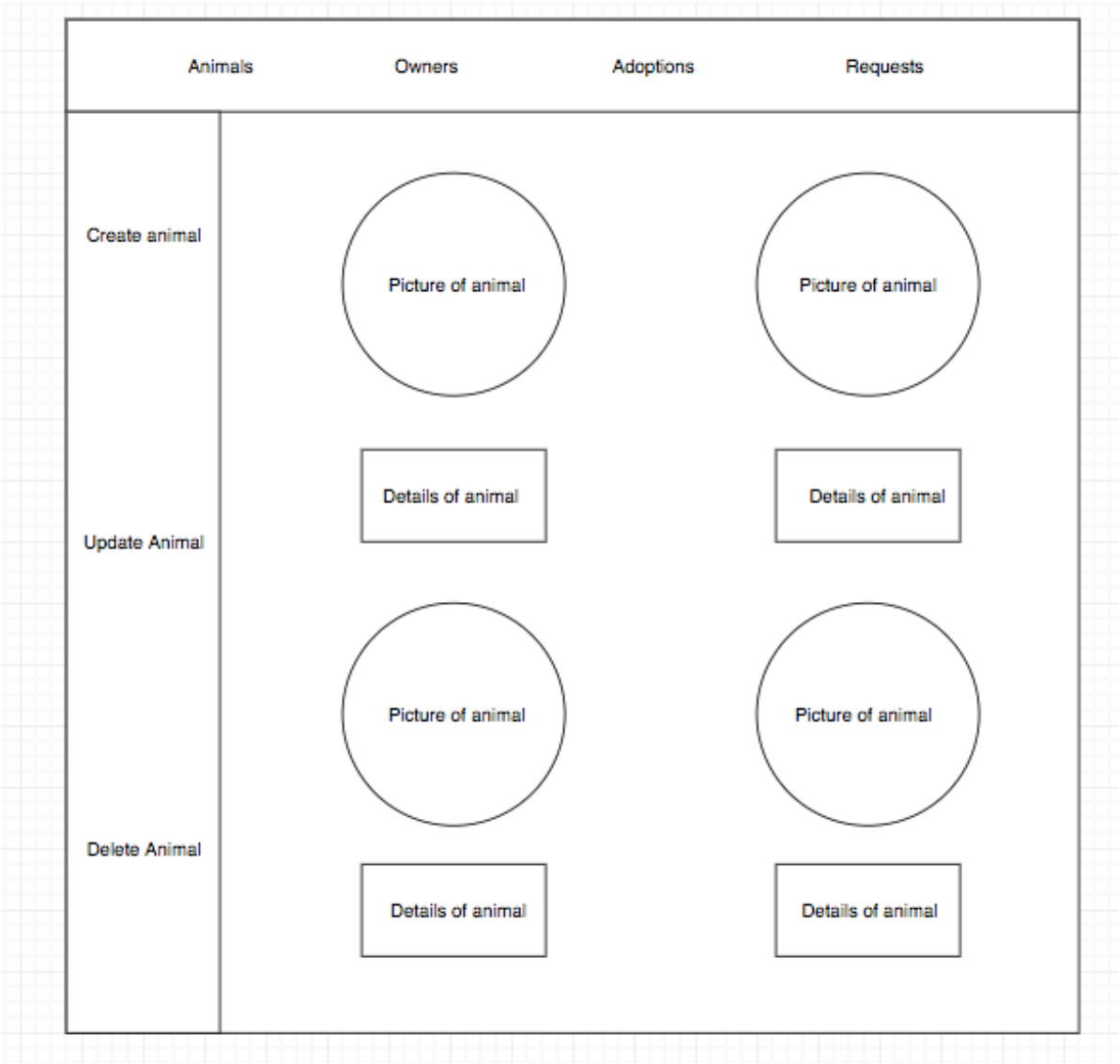


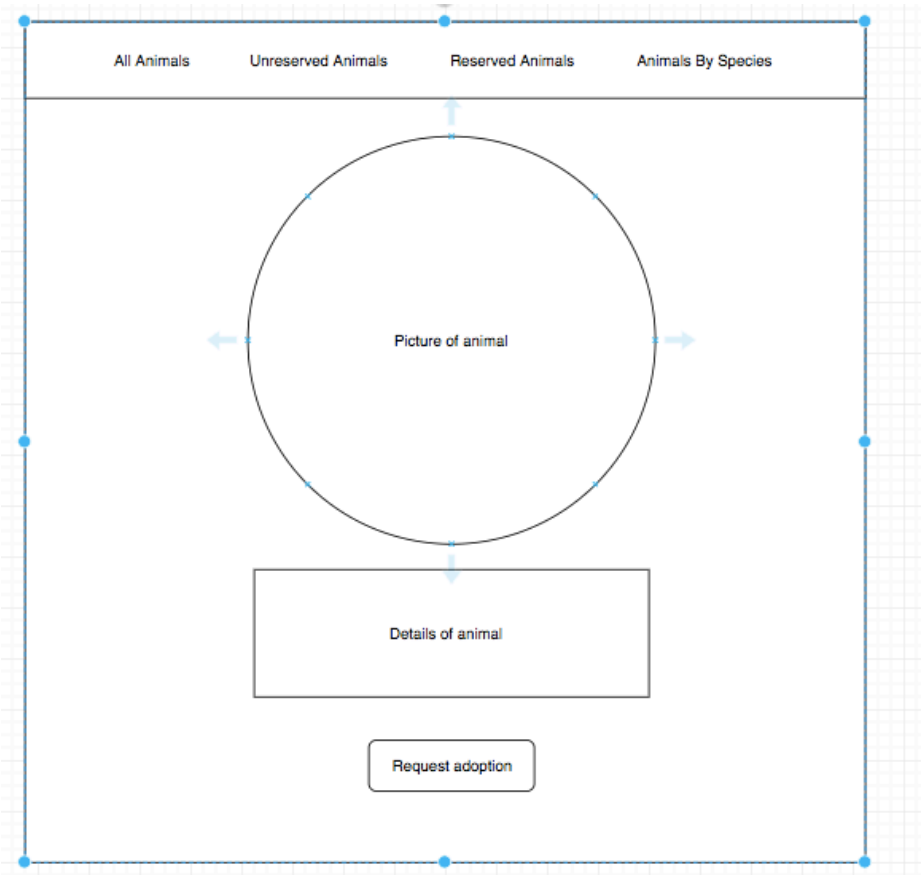
Description here

This User Site Map shows a comic book databases. Users can log in, view comics and characters and publishers. They can also leave reviews on comics and interlink between related detail.

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams

Paste Screenshot here





Description here

Above are two wireframe diagrams. The first shows the manager user page where they can navigate and view all animals.

The second shows a single animal page viewed by potential adopter. They can requests an adoption from this page.

Week 5

Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method

Paste Screenshot here

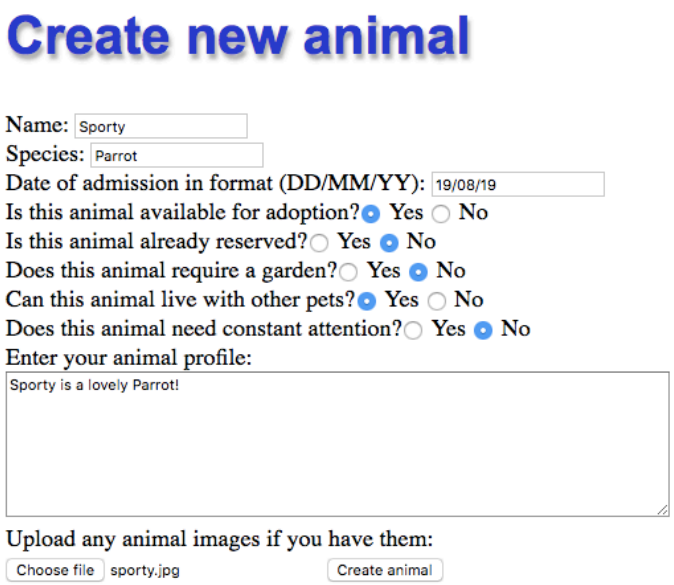
```ruby
# This method will allow users to post their images when submitting new animal details
post '/manager/animals' do
  # An @animal object is first assigned to the entered details
  @animal = Animal.new(params)
  # A filename variable is then assigned to the file name of the image uploaded
  @filename = params[:image][:filename]
  # A filepath variable is assigned to the desired filepath where the image will be saved
  @filepath = "/images/#{@filename}"
  # A file variable is assigned to the image file that is temporarily accessible from the upload
  file = params[:image][:tempfile]
  # The file is then read(essentially copied) and then wrote(pasted) into its new folder directory
  File.open("./public/images/#{@filepath}", 'wb') do |f|
    f.write(file.read)
  end
  # The animal image key is then accessed in the ruby animal object and the value is then assigned to the storage filepath
  @animal.image = @filepath
  # The animal object is then called upon by an appropriate method which will log the deails in an sql
  # database
  @animal.save()
  # The erb browser address for this method is assigned.
  erb (:"manager/new_animal")
end
```

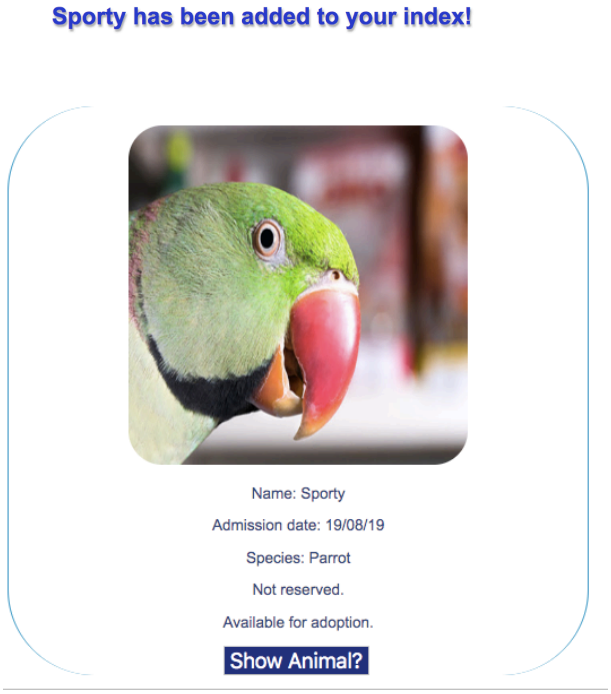
Description here

This interactive ruby method allows an image file to be uploaded to the application. The pseudocode explains what each line of the code is supposed to do.

Unit	Ref	Evidence
P	P.13	Show user input being processed according to design requirements. Take a screenshot of: * The user inputting something into your program * The user input being saved or used in some way

Paste Screenshot here



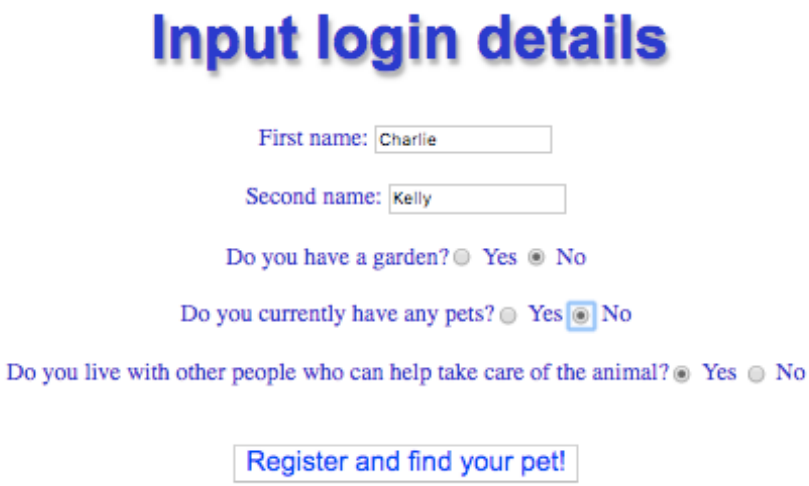


Description here

These photos show a new animal being created by the user, a shelter manager. On a successful submit they then see confirmation of their details having been registered. Finally, we see the successful result on their index page.

Unit	Ref	Evidence
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved

Paste Screenshot here



Thank you Charlie Kelly! Click below to view all animals.

Click to view animals

```
shelter=# select * from owners;
 first_name | second_name | has_garden | has_no_pets | lives_with_others | id
------------+-------------+------------+-------------+-------------------+----
 Charlie    | Kelly       | f          | f           | t                 | 23
```

Description here

Above a user, a prospective adopter, logs their details. They are then taken to a confirmation page of their details being saved. Finally, we can see on sql that their details have been successfully saved.

Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program

Paste Screenshot here

Name: Patch

Admission date: 22/03/19

Species: Dog

Not reserved.

Not available

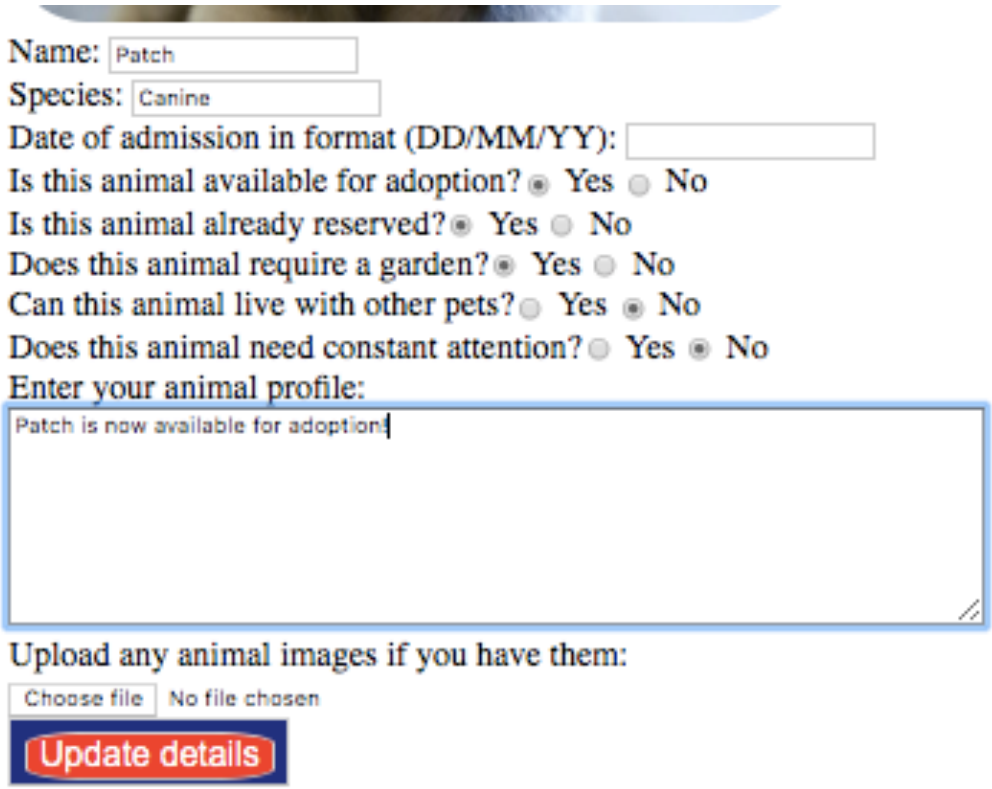
Animal does not need a garden.

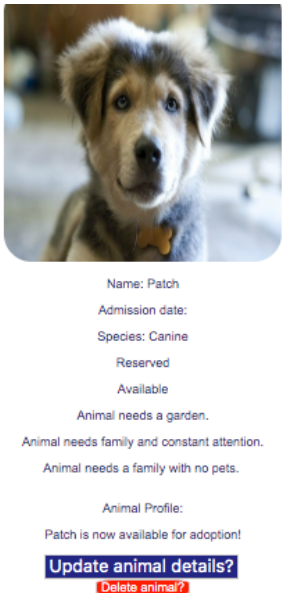
Animal is okay alone.

Animal is okay with other pets.

Animal Profile:

Update animal details?





Description here

The above screenshots show the original state of an animal profile, the owner then editing the details and the updated profile.

Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.

Paste Screenshot here

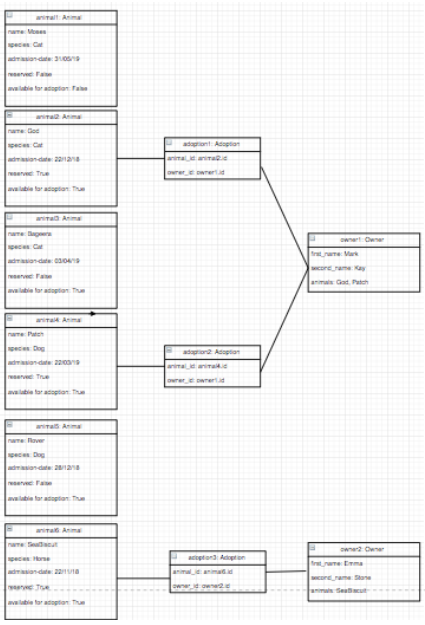
https://github.com/shaunhk/shelter

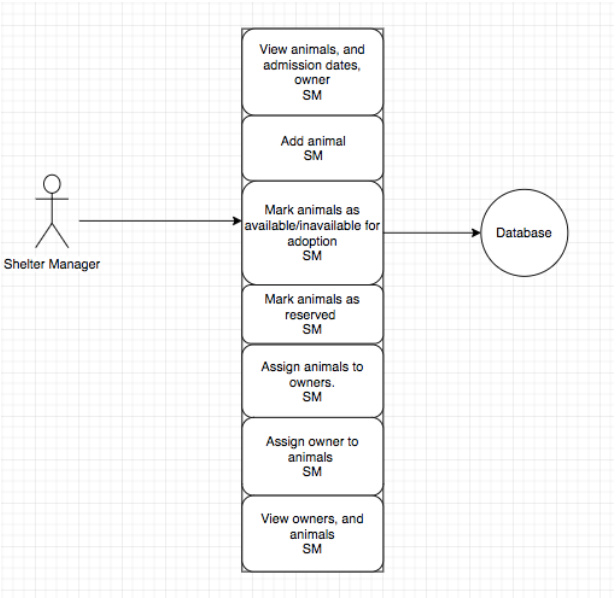
Description here

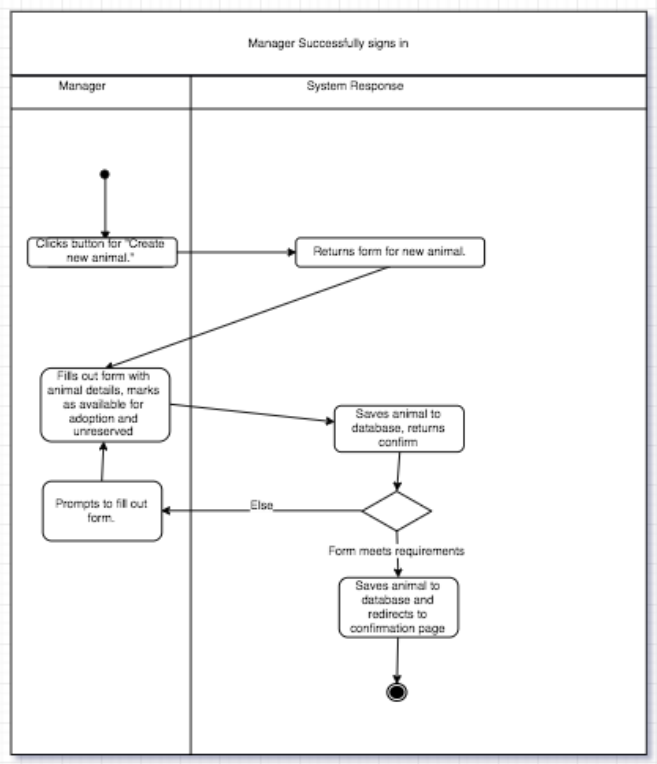
Above is a project called Petsmatch, an app which allows shelter managers to manage their animals and also potential owners to find suitable animals and make requests for adopting them.

Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.

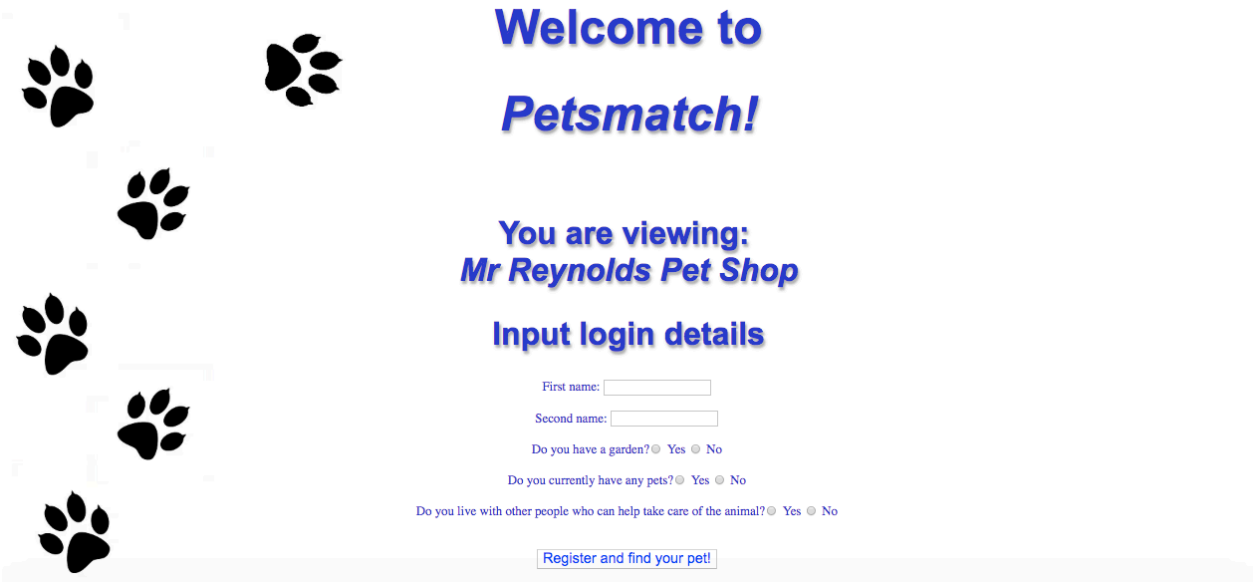
Paste Screenshot here







Topic	Possible effect and constraint on Product	Possible solution
Hardware and software constraints	Limited ram and processing power	Shouldn't affect simple app protoype
Performance requirements	Low performance requirements, simple web app	Shouldn't be an issue
Server problems	Potentially unlimited traffic	A select product for animal shelter managers, however if expanded to include potential owners, this could be an issue. Expand database, monitor growth and react
Usability	Visually impaired users, not technically proficient users	Include alt tags in html and use reasonably sized images. Have big tags for navigating site, use reasonable font sizes.
Budgets	No money	Shouldn't be an issue developing first prototype
Time limitations	Deadline 05/06/2019	Effective time management and allow time for debugging, UX adjustments and error.



Description here

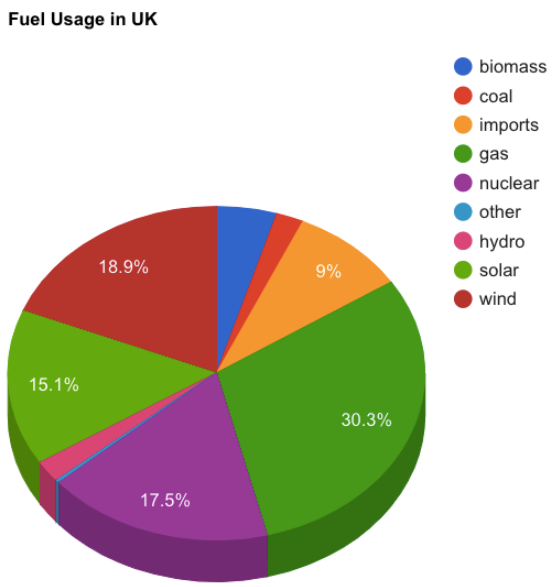
Above is an object diagram, a use case diagram, an activity diagram, an implementations and constraints diagram and examples of development in my app.

Week 7

Unit	Ref	Evidence
P	P.16	Show an API being used within your program. Take a screenshot of: * The code that uses or implements the API * The API being used by the program whilst running

Paste Screenshot here

```
data(){
  return {
    fuelTypes: null,
    fuelIntensity: null,
    chartOptionsUK: {
      title: 'Fuel Usage in UK',
      subtitle: 'Fuel types and percentage used in UK',
      is3D: true
    },
    fuelIntensityOptions: {
      title: 'Intensity of Fuel Usage per Fuel Type',
      subtitle: '',
      is3D: true
    }
  }
},
mounted() {
fetch('https://api.carbonintensity.org.uk/intensity/factors')
.then(response => response.json())
.then(trams => this.fuelIntensity = trams.data),
fetch('https://api.carbonintensity.org.uk/generation')
.then(response => response.json())
.then(data => this.fuelTypes = data.data.generationmix)
},
components: {
  GChart
}
```



Description here

The above images show an api fetch request being called in the inbuilt mounted function of Vue.js. The data function is used to return this information in a user friendly pie chart, one of which is also shown.

Week 8

Unit	Ref	Evidence
P	P.2	Take a screenshot of the project brief from your group project.

Paste Screenshot here

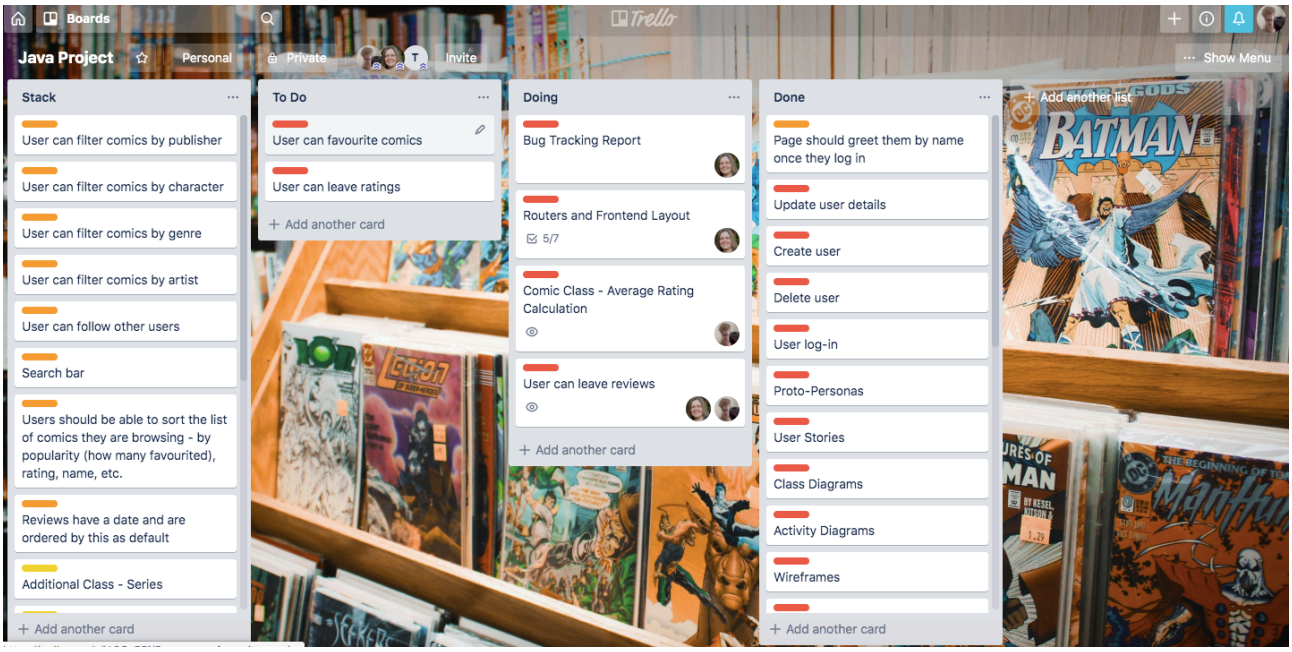
```
# Educational App

The BBC are looking to improve their online offering of educational content by
developing some interactive browser applications that display information in a fun and
interesting way. Your task is to make an a Minimum Viable Product or prototype to put
forward to them - this may only be for a small set of information, and may only
showcase some of the features to be included in the final app.

## MVP

A user should be able to:

- view some educational content on a particular topic
- be able to interact with the page to move through different sections of content

## Example Extensions

- Use an API to bring in content or a database to store information.
- Use charts or maps to display your information to the page.

## API, Libraries, Resources

- https://www.highcharts.com/ HighCharts is an open-source library for rendering
responsive charts.
- https://leafletjs.com/ Leaflet is an open-source library for rendering maps and map
functionality.
```

Description here

Above is a project brief for an educational app we made as a group.

Unit	Ref	Evidence
P	P.3	Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.

Paste Screenshot here



Description here

Above is a trello board used to plan and manage tasks as a group when building a comic database app.

Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.

Paste Screenshot here

Java Project **Acceptance Criteria & Test Plan**

Acceptance Criteria	Expected Result	Pass/Fail
The comic book lover is able to create a profile	The site allows a new user to create a profile when they select the 'Register' button.	Pass
The comic book lover is able to log-in to the site	The site allows entry to the homepage when the comic book lover logs in.	Pass
The comic book lover is able to view their profile	The site allows the comic book lover to view all their details when they click on the user-profile button.	Pass
The comic book lover is able to update their profile details	The site provides the comic book lover with a pre-populated form for them to change any details as necessary, when they click on an 'Edit Profile' button. It then saves any changes.	Pass
The comic book lover is able to delete their profile	The site will delete a profile from the site when the 'Delete' button is clicked.	Pass
The comic book lover is able to leave a review on a comic	The site will allow the user to leave a review and a rating on the comic.	Fail
The comic book lover is able to make a comic a 'favourite'.	The site will mark a comic as a favourite against a profile when the 'Favourite' button is selected.	Fail

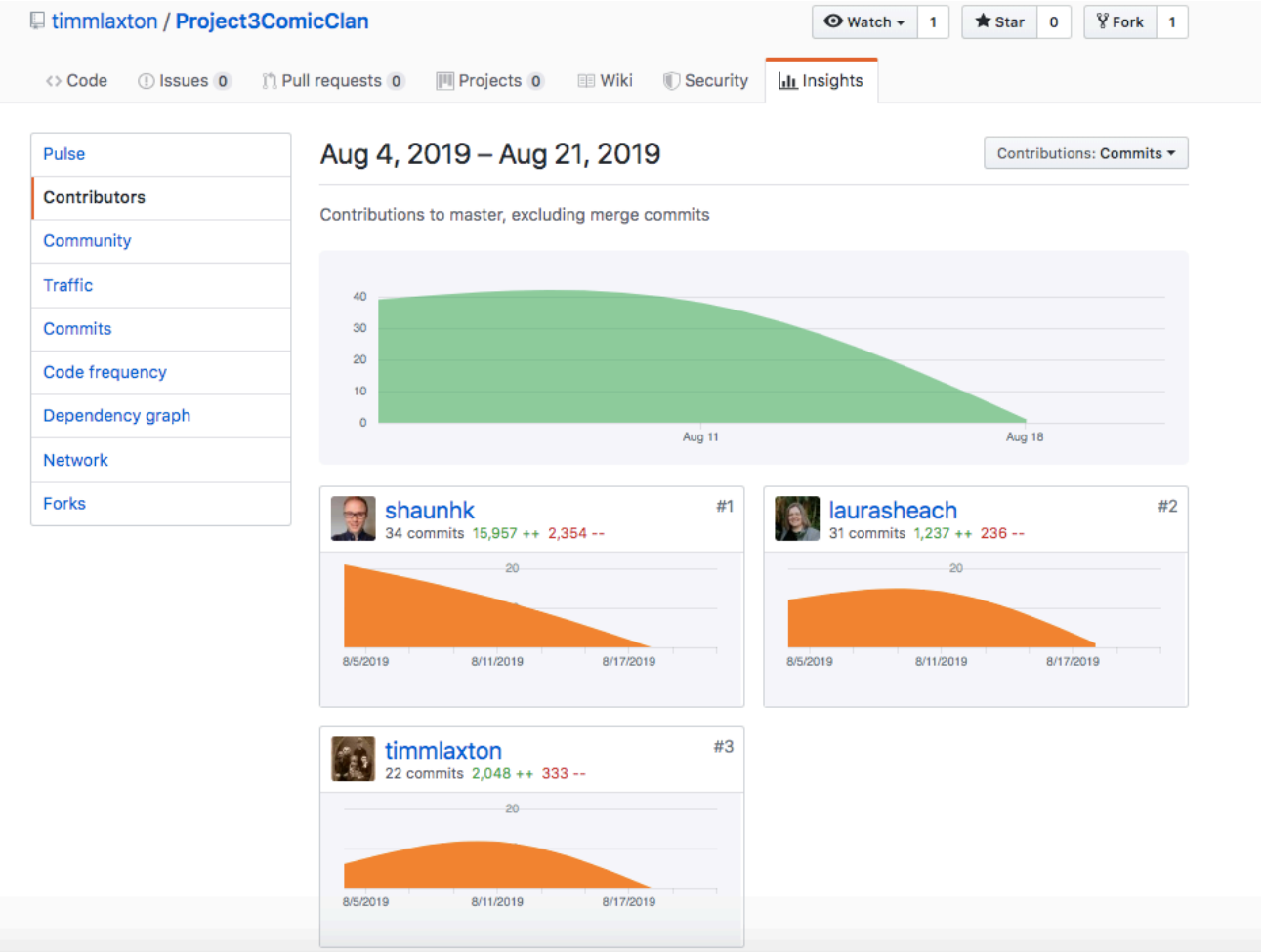
Description here

Above is an acceptance criteria and test plan for our app, showing accepted results and whether we met them by our first deadline.

Week 9

Unit	Ref	Evidence
P	P.1	Take a screenshot of the contributor's page on Github from your group project to show the team you worked with.

Paste Screenshot here



Description here

Above is our Github contributors' page. It shows my teammates and me (shaunhk) and our contributions.

Week 11

Unit	Ref	Evidence
P	P.18	Demonstrate testing in your program. Take screenshots of: * Example of test code * The test code failing to pass * Example of the test code once errors have been corrected * The test code passing

Paste Screenshot here

Git × | testing_ta... | testing_ta... | spec_test... | testing_ta... | Static_&_... | card.rb

```ruby
require_relative('../testing_task_2.rb')
require('minitest/autorun')
require('minitest/rg')
require_relative('../card.rb')

class TestCard < MiniTest::Test

  def setup
    @ace = Card.new("spades", 1)
    @two_hearts = Card.new("hearts", 2)
    @king = Card.new("spades", 10)
    @cards = [@ace, @two_hearts, @king]
    @game = CardGame.new(@cards)
  end

  def test_card()
    assert_equal(@game.checkforAce(@ace), true)
  end
  #
  def test_card_false()
    assert_equal(@game.checkforAce(@king), false)
  end

  def test_highest_card
    assert_equal(@game.highest_card(@king, @two_hearts), @king)
  end

  def test_cards_total()
    assert_equal(@game.cards_total(@cards), "You have a total of 13")
  end


end
```

```
specs ruby spec_test_task_1.rb
spec_test_task_1.rb:1:in `require_relative': /Users/Shaunhk/Desktop/CodeClan/course_work/
week_5_pda/PDA_Static_and_Dynamic_Task_A/testing_task_2.rb:25: syntax error, unexpected k
eyword_end, expecting end-of-input (SyntaxError)
        from spec_test_task_1.rb:1:in `<main>'
specs
```

Git | testing_ta... | testing_ta... | spec_test... | testing_ta... | Static_&_... | card.rb

```ruby
require_relative('../testing_task_2.rb')
require('minitest/autorun')
require('minitest/rg')
require_relative('../card.rb')

class TestCard < MiniTest::Test

  def setup
    @ace = Card.new("spades", 1)
    @two_hearts = Card.new("hearts", 2)
    @king = Card.new("spades", 10)
    @cards = [@ace, @two_hearts, @king]
    @game = CardGame.new(@cards)
  end

  def test_card()
    assert_equal(@game.checkforAce(@ace), true)
  end
  #
  def test_card_false()
    assert_equal(@game.checkforAce(@king), false)
  end

  def test_highest_card
    assert_equal(@game.highest_card(@king, @two_hearts), @king)
  end

  def test_cards_total()
    assert_equal(@game.cards_total(@cards), "You have a total of 13")
  end


end
```

```
specs ruby spec_test_task_1.rb
spec_test_task_1.rb:1:in `require_relative': /Users/Shaunhk/Desktop/CodeClan/course_work/
week_5_pda/PDA_Static_and_Dynamic_Task_A/testing_task_2.rb:25: syntax error, unexpected k
eyword_end, expecting end-of-input (SyntaxError)
        from spec_test_task_1.rb:1:in `<main>'
specs ruby spec_test_task_1.rb
Run options: --seed 47264

# Running:

....

Finished in 0.001540s, 2597.4025 runs/s, 2597.4025 assertions/s.

4 runs, 4 assertions, 0 failures, 0 errors, 0 skips
specs 4
```

Description here

Above there is test code and the test results shown after the code has been fixed.

Unit	Ref	Evidence
I&T	I.T.1	The use of Encapsulation in a program and what it is doing.

Paste Screenshot here

```java
public class Passenger {

    private String name;
    private int bags;
    private Flight flight;
    private int seatNumber;

    public Passenger(String name, int bags){
        this.name = name;
        this.bags = bags;
        this.flight = null;
        this.seatNumber = 0;
    }

    public String getName() { return this.name; }

    public int getBags() { return this.bags; }

    public Object getFlight() { return this.flight; }

    public void setName(String name) {
        this.name = name;
    }

    public void setBags(int bags) {
        this.bags = bags;
    }

    public void setFlight(Flight flight) {
        this.flight = flight;
    }

    public void setSeatNumber(int seatNumber) {
        this.seatNumber = seatNumber;
    }
```

Description here

Above is an example of encapsulation for a java class with private variables and appropriate getters and setters.

Week 12

Unit	Ref	Evidence
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.

Paste Screenshot here

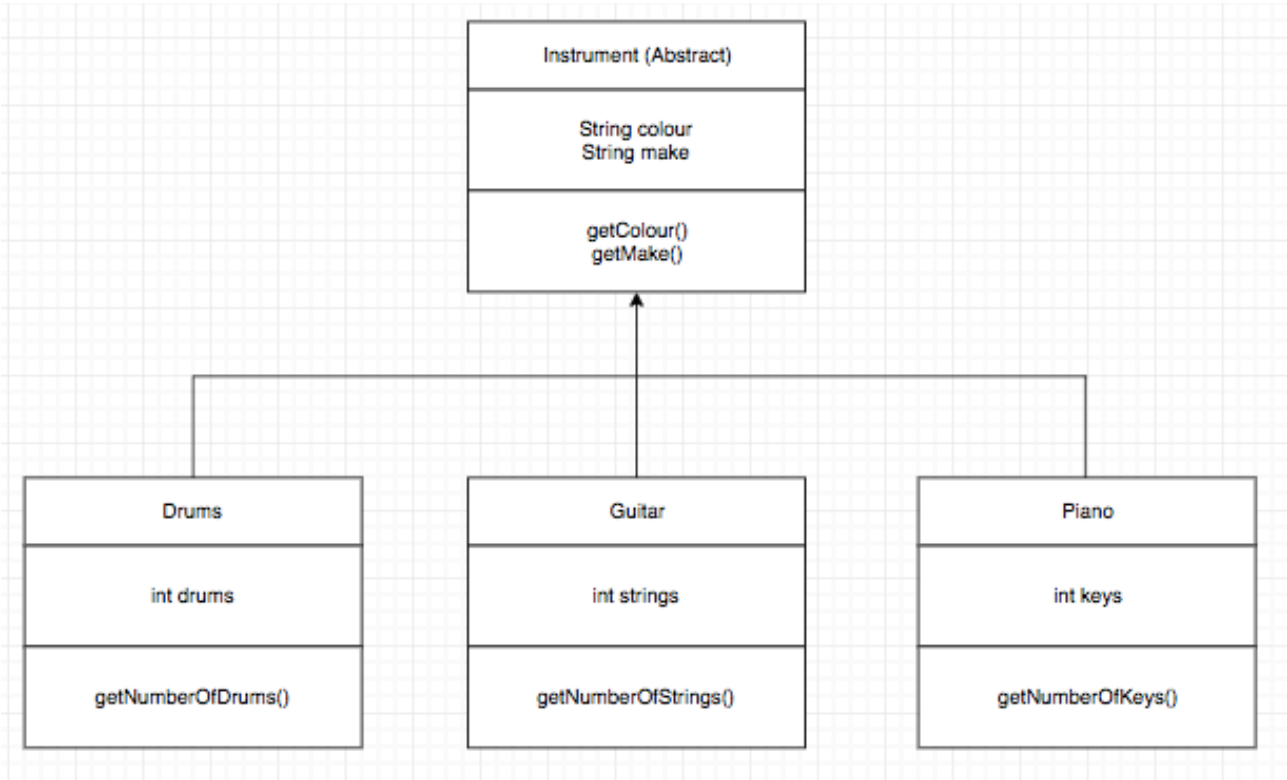
```java
package Instruments;

import Behaviours.IPlay;
import Instruments.InstrumentsEnums.InstrumentColour;
import Instruments.InstrumentsEnums.InstrumentMake;
import OtherAbstractClasses.Sell;

public abstract class Instrument extends Sell implements IPlay {

    private InstrumentColour colour;
    private InstrumentMake make;

    public Instrument(double sellingPrice, double buyingPrice, InstrumentColour colour, InstrumentMake make) {
        super(sellingPrice, buyingPrice);
        this.colour = colour;
        this.make = make;
    }

    public InstrumentColour getColour() { return colour; }

    public InstrumentMake getMake() { return make; }
}
```

Description here

Above is an abstract class 'Instrument' which inherits from the abstract class Sell and also implements the interface 'IPlay'. It ensures that every class that inherits from 'Instrument' must have the declared variables of colour and make, as well as the variables 'sellingPrice' and 'buyingPrice' inherited from 'Sell'. All classes that inherit from 'Instrument' will have the methods 'getColour' and 'getMake' available to them. Both the abstract class 'Instrument' itself and any classes that inherit from it will require concrete examples of any methods specified inside the interface IPlay.

Unit	Ref	Evidence
A&D	A.D.5	An Inheritance Diagram

Paste Screenshot here



Description here

Above shows a simple inheritance diagram in which three classes ‘Drums’, ‘Guitar’ and ‘Piano’ inherit from the abstract class instrument.

Unit	Ref	Evidence
I&T	I.T.2	Take a screenshot of the use of Inheritance in a program. Take screenshots of: *A Class *A Class that inherits from the previous class *An Object in the inherited class *A Method that uses the information inherited from another class.

Paste Screenshot here

```
public abstract class Components {

    private String make;
    private String model;

    public Components(String make, String model){
        this.make = make;
        this.model = model;
    }

    public String getMake() { return this.make; }

    public String getModel() { return this.model; }

}
```

```
public class Radio extends Components{

    private String tune;

    public Radio(String model, String make){
        super(model, make);
        this.tune = tune;
    }

    public void tune(String station) { this.tune = station; }

    public String getPlaying() { return this.tune; }
}
```

```
radio = new Radio( model: "AS61",  make: "Toshiba");
```

```
radio.getMake();
```

Description here

In the above images we can see an abstract class ‘Components’, a ‘Radio’ class which inherits from ‘Components’, A radio object of the Radio class and finally the object calling with the inherited ‘getMake( )’ method

Week 14

Unit	Ref	Evidence
P	P.9	Select two algorithms you have written (NOT the group project). Take a screenshot of each and write a short statement on why you have chosen to use those algorithms.

Paste Screenshot here

```
Park.prototype.dietTypes = function(){
  const diet = {'carnivore': 0, 'herbivore': 0, 'omnivore': 0}
  for (dinosaur of this.dinosaurs){
    if (dinosaur.diet === 'carnivore'){
      diet.carnivore += 1
    }else if (dinosaur.diet === 'herbivore'){
      diet.herbivore += 1
    }else if (dinosaur.diet === 'omnivore'){
      diet.omnivore += 1
    }
  }
  return diet
}
```

```
public void getAverageRating() {
    double total = 0;
    for (int i = 0; i < this.getReviewsAmount(); i++) {
        total += this.reviews.get(i).getRating();
        double averageRating = total / this.getReviewsAmount();
        this.rating = averageRating;
    }

}
```

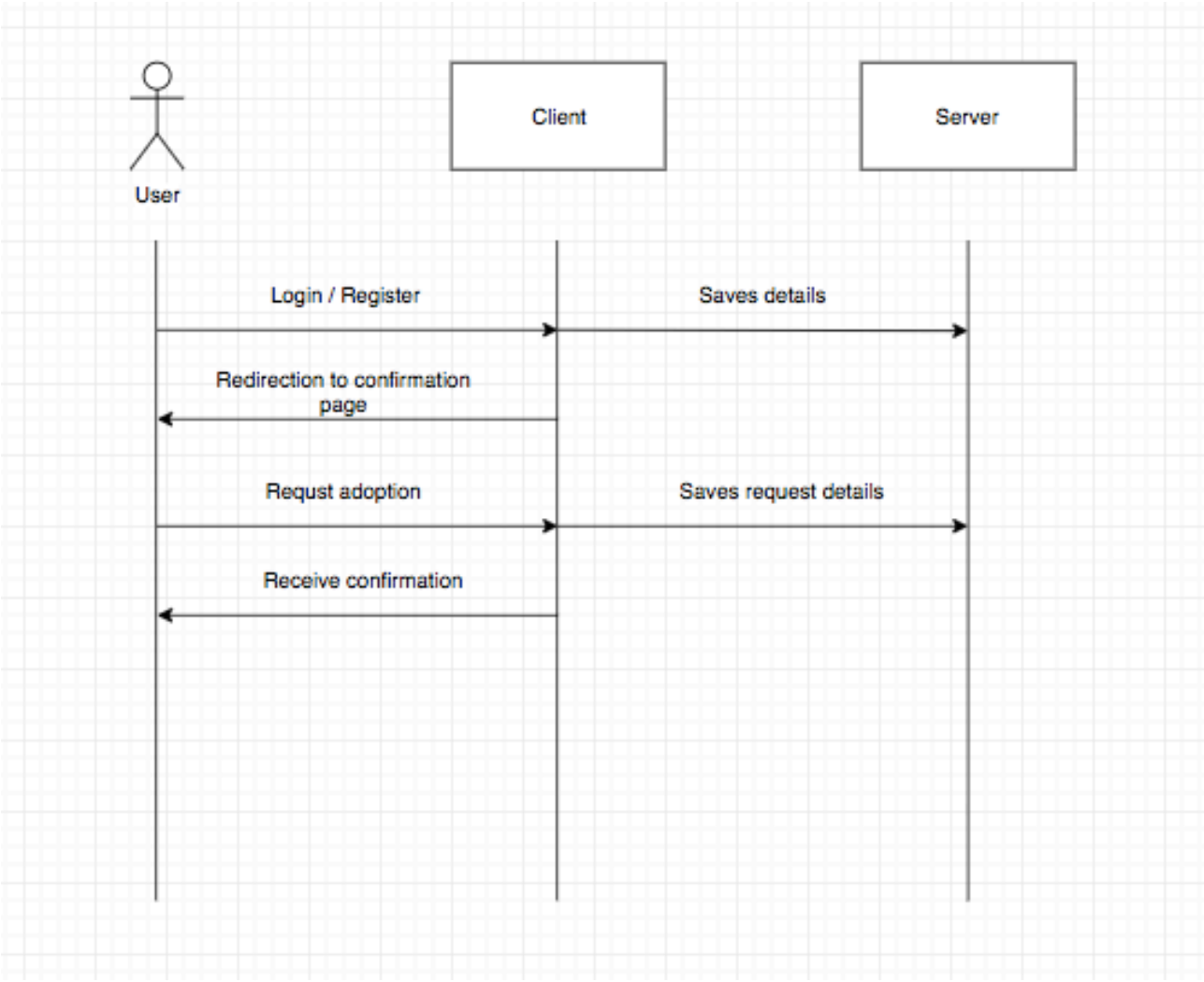
Description here

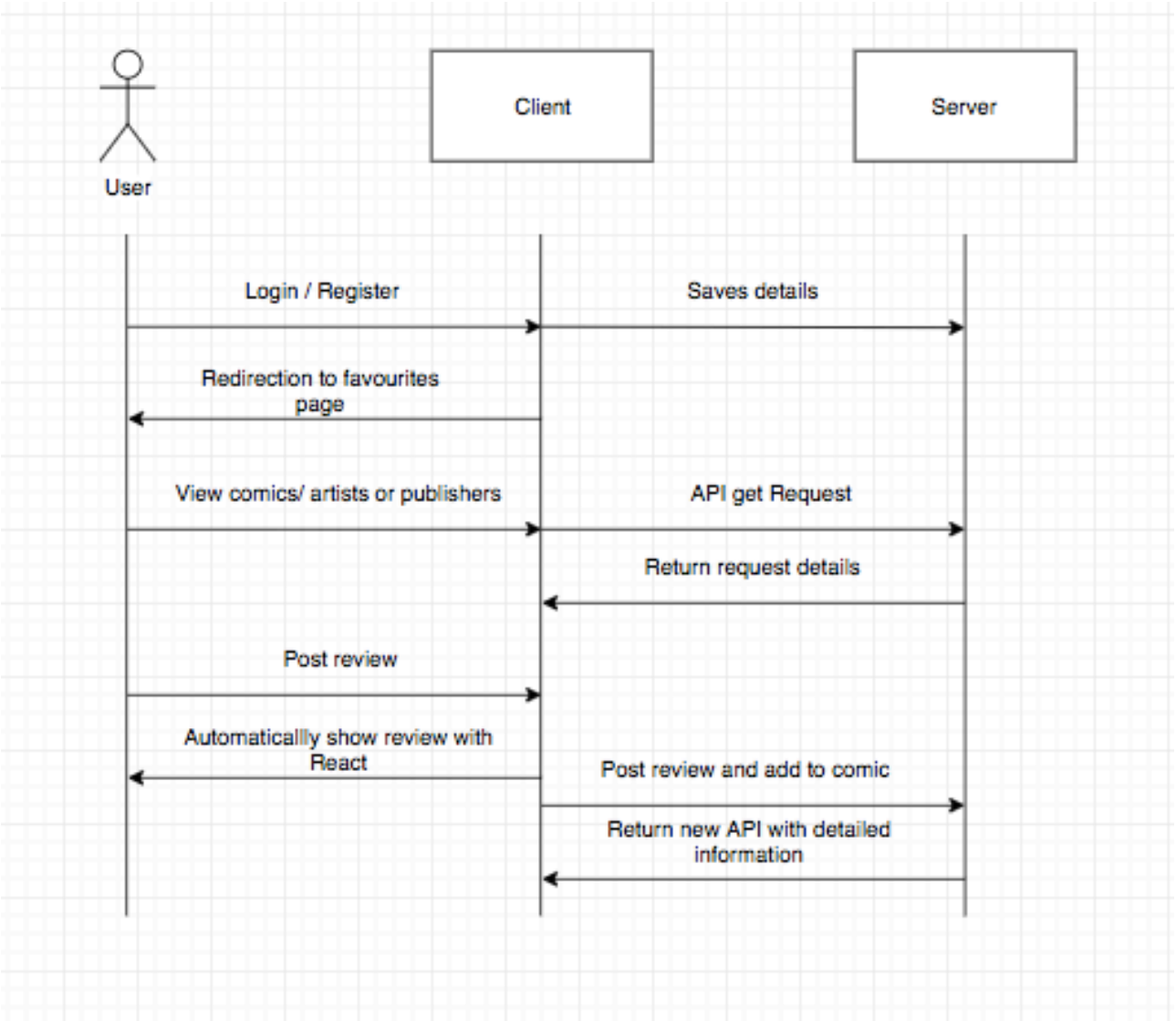
The first javascript algorithm is used to access an array of dinosaurs contained in a Park object. Each dinosaur is an object of the class Dinosaur with a fixed diet string equal to one of three possibilities. The algorithm sets up a hash to be returned with the number of dinosaurs with the diet as a value integer corresponding to each key String 'diet'. It thus returns the number of dinosaurs in the park with each diet type.

The second java algorithm is for calculating the average review for a comic in our group project by accessing the reviews array (which is made up of Review objects) and accessing their rating, adding each one to a total. This total is then divided by the size of the array, giving an average rating.

Unit	Ref	Evidence
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).

Paste Screenshot here



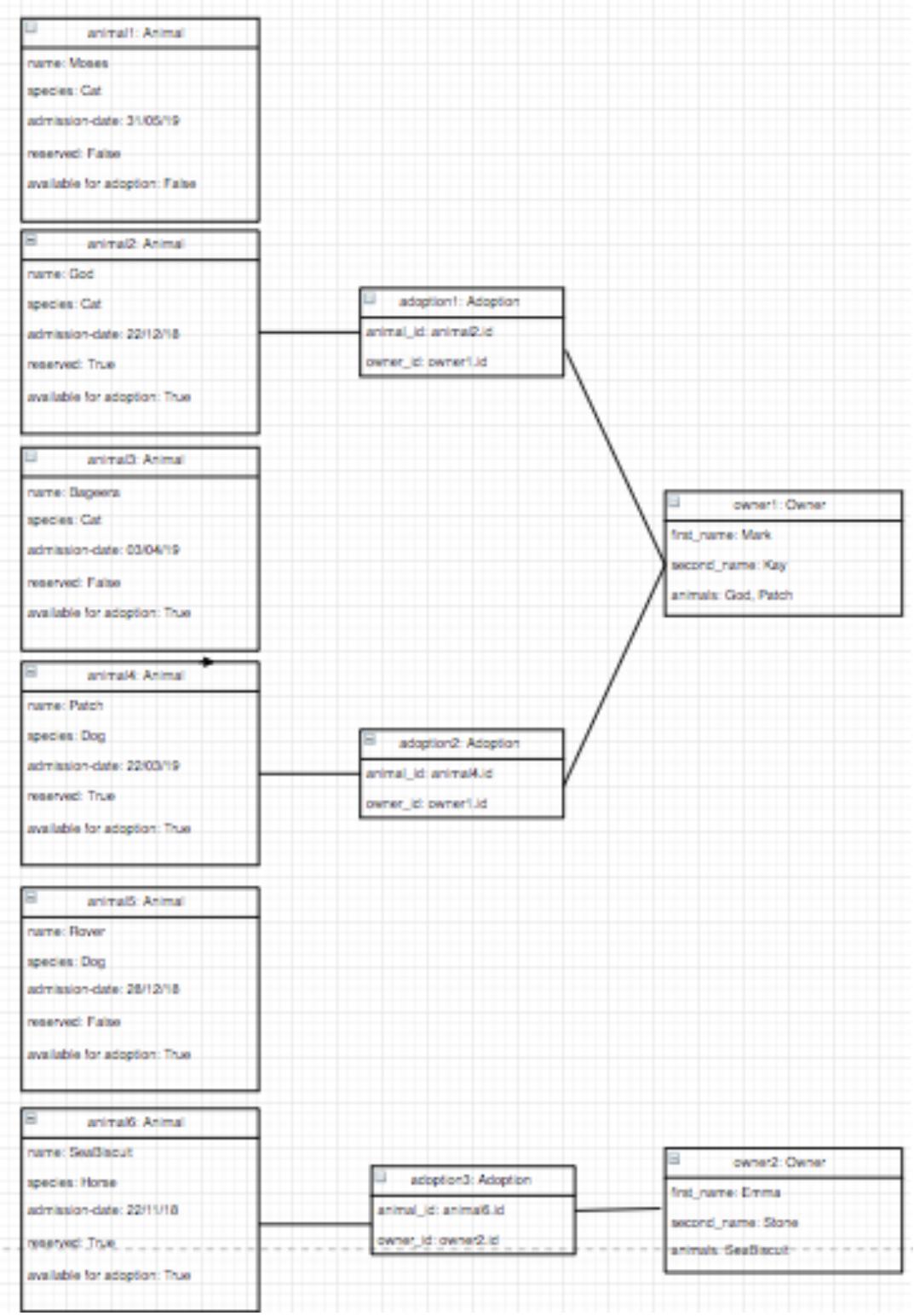


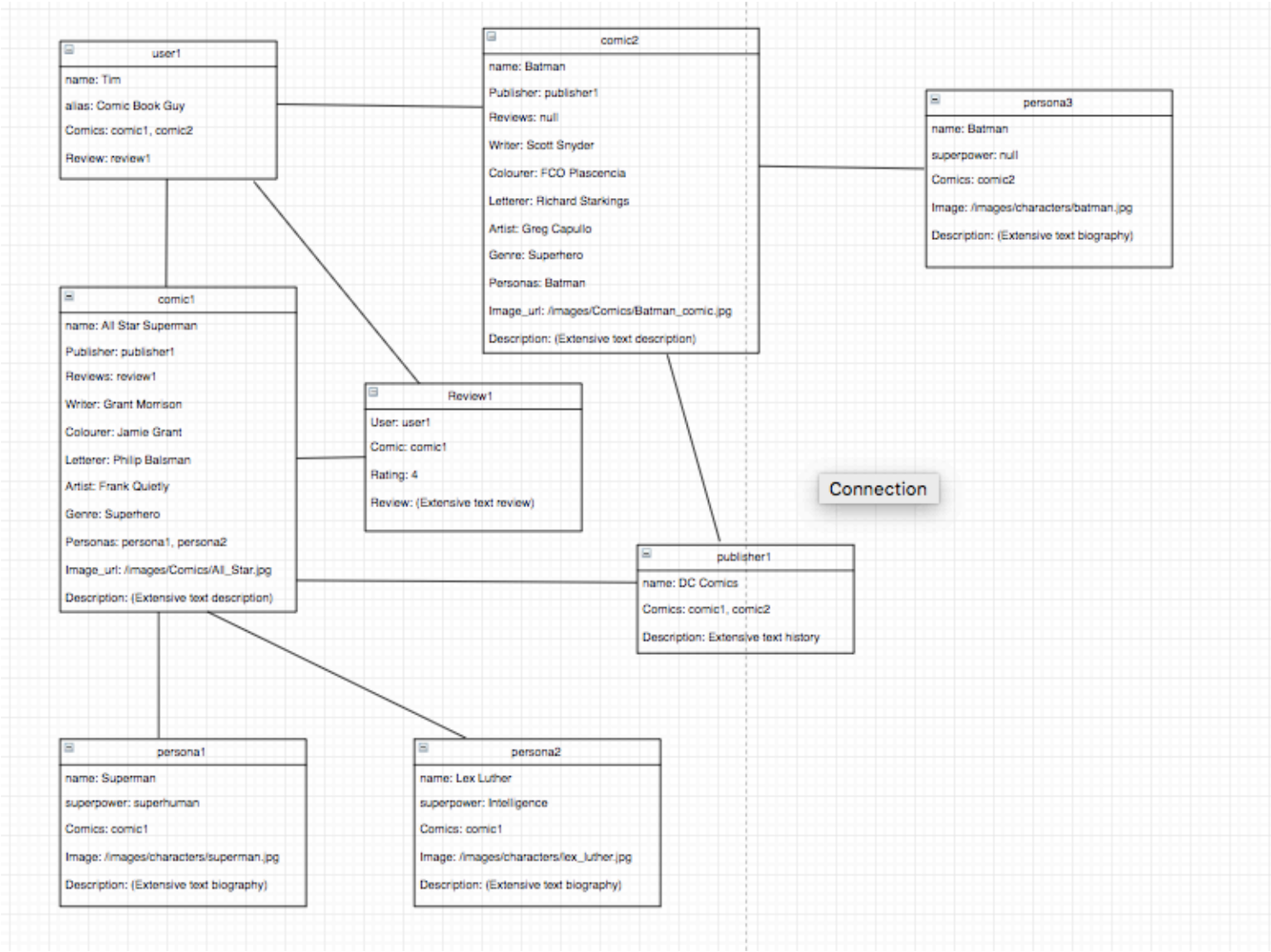
Description here

Above are two simple system interaction diagrams. They show the direction of information passed between a user a local client and server, in which a database of information is stored.

Unit	Ref	Evidence
P	P.8	Produce two object diagrams.

Paste Screenshot here





Description here

Above are two object diagrams. The first is from the solo ‘Petsmatch’ project, the second is from the group comics project.

Unit	Ref	Evidence
P	P.17	Produce a bug tracking report

Paste Screenshot here

Java Group Project **Bug Tracking Report**

Bug/Error	Solution	Date
Comic Vine API - CORS error & therefore could not fetch the information from the API as required.	Could not get around this issue, and therefore decided to create our own to use for our app.	8th August 2019
Could not connect the database tables	The original code was using an ArrayList<> type the the variables were declared, however this does not work with SQL on join columns and tables - needs to be List<>	9th August 2019
Could not create a new profile for a user	Had to refactor the backend code to correct the file paths to allow the frontend and backend to connect with one another.	12th August 2019
Frontend could not hold onto the id of a user on login	Had to refactor the frontend code, changing a stateless component to a class, allowing a series of functions to be created to store the user id and remain logged in for that user.	12th August 2019
Frontend could not get a hold of the id in the database for a new user and keep them logged in	Following the post part of the function to the database, we were able to capture the response from the database and attach it to the new user information and feed that into the function that held onto the current user, allowing them to login.	13th August 2019
Could not follow the paths correctly in the frontend to get the right information out of the database where the tables joined.	Refactored the projections code to provide the right paths to pull out the correct information to display.	14th August 2019

Description here

Above is a bug tracking report from our group comics project. It shows relative bugs, solutions and dates for the occuring bugs.