

WEEK 2.

Demonstrate the use of an array in a program.

- An Array
- The function that uses the array.
- The result of the function running.

The screenshot shows a code editor with two windows. The left window is a terminal with the command `day_01 ruby array.rb` and its output: `["this", "is", "an", "array", "of", "strings"]` and `["strings", "of", "array", "an", "is", "this"]`. The right window shows the `array.rb` file with the following code:

```
1 array = Array.new
2
3 array.push("this", "is", "an", "array", "of", "strings")
4
5 p array
6
7 array_reversed = array.reverse!()
8
9 p array_reversed
10
```

Demonstrate the use of hash in a program.

- The Hash.
- The function using hash.
- The result of the function.

The screenshot shows a code editor with two windows. The left window is a terminal with the command `day_01 ruby hash.rb` and its output: `"PDA's deadline is on the 26/10/2018."`. The right window shows the `hash.rb` file with the following code:

```
1 hash_project = {
2   title: "PDA",
3   deadline: "26/10/2018",
4   subjects: ["Ruby", "Java", "JavaScript"]
5 }
6
7 hash_project[:other_technologies] = ["Sinatra", "Hibernate",
8   "React"]
9
10 p "#{hash_project[:title]}s deadline is on the
11   #{hash_project[:deadline]}."
```

WEEK 3.

Demonstrate searching data in a program.

- Function that searches data.
- The result of the function running.

```
def plants_pollinated_by_bees()
  sql = "SELECT plants.* FROM plants INNER JOIN food ON food.plant_id = plants.id WHERE food.bee_id = $1;"
  values = [@id]
  results = SqlRunner.run(sql, values)
  return results.map {|plant| Plant.new( plant )}
end
```

Lavender

Season to plant:

Autumn

Pollen:

10

Pollinated by:

Digory

Joanna

CHANGE MY PLANT

DELETE

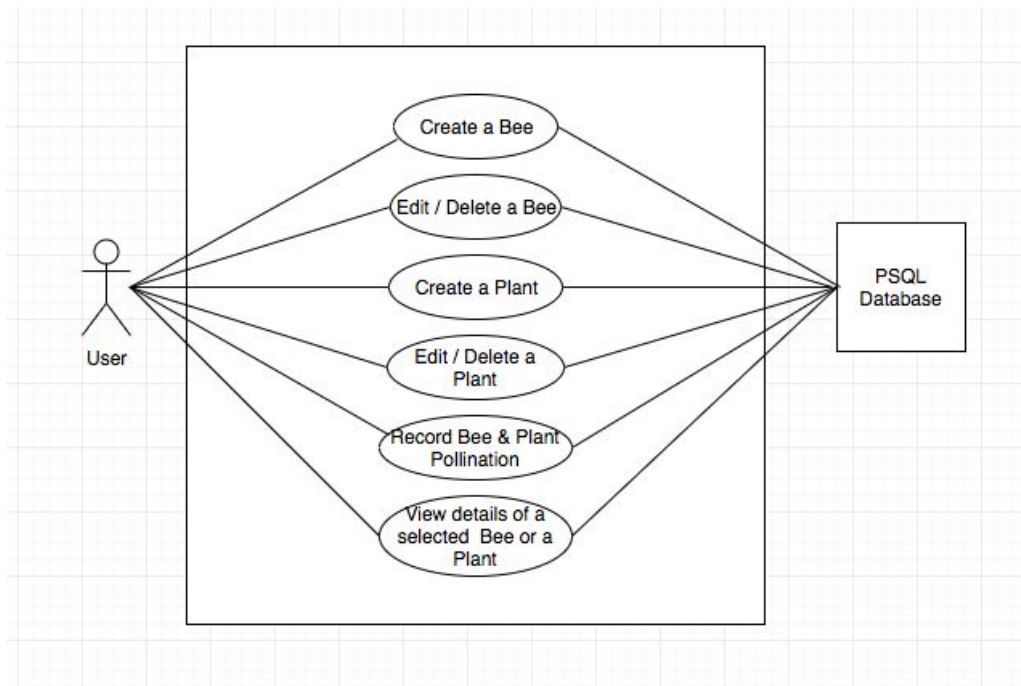
Demonstrate sorting data in a program.

- Function that sorts data.
- The result of the function running.

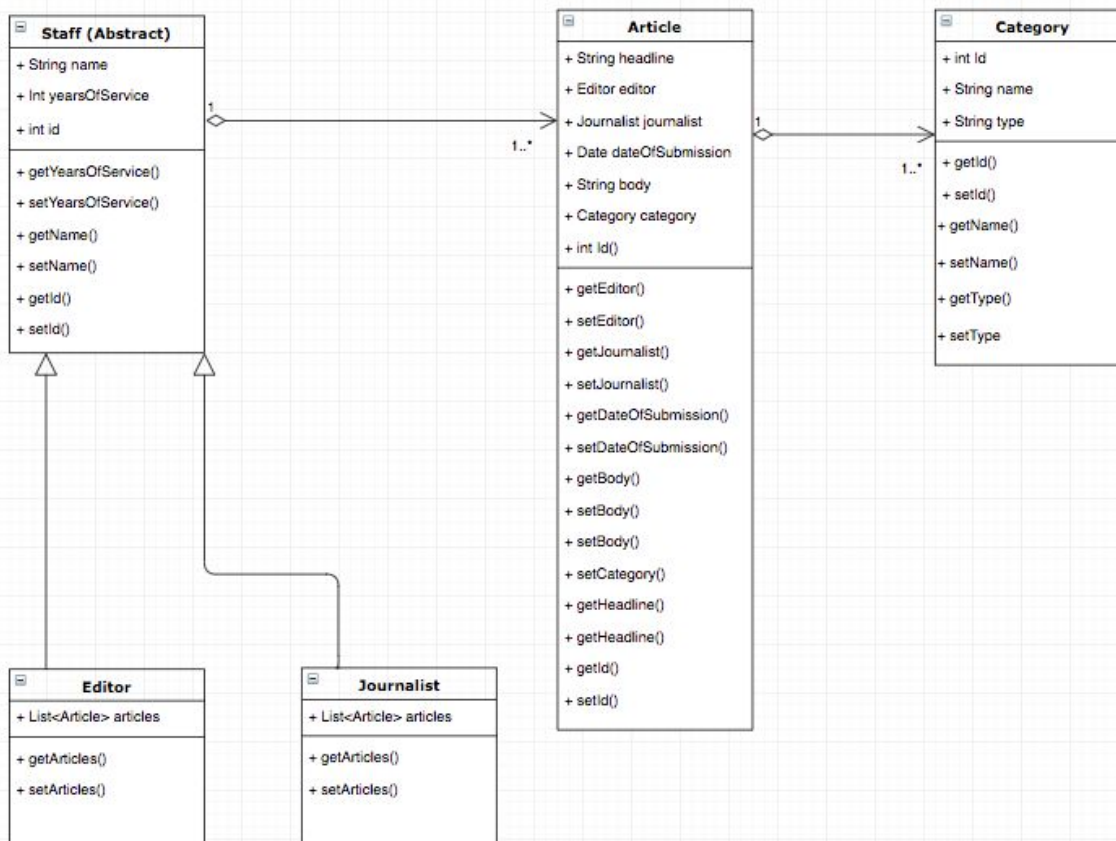
```
def self.most_popular()
  sql = "SELECT screening_id,
  COUNT(*) AS count
  FROM tickets
  GROUP BY screening_id
  ORDER BY count
  DESC LIMIT 1;"
  most_popular_id = SqlRunner.run(sql).first["screening_id"].to_i
  Screening.find(most_popular_id)
end
```

WEEK 5 AND 6.

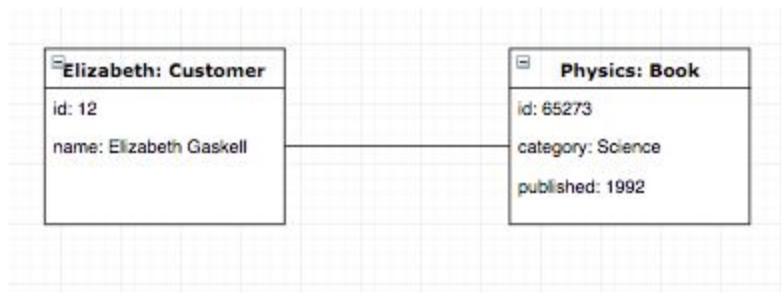
A Use Case Diagram



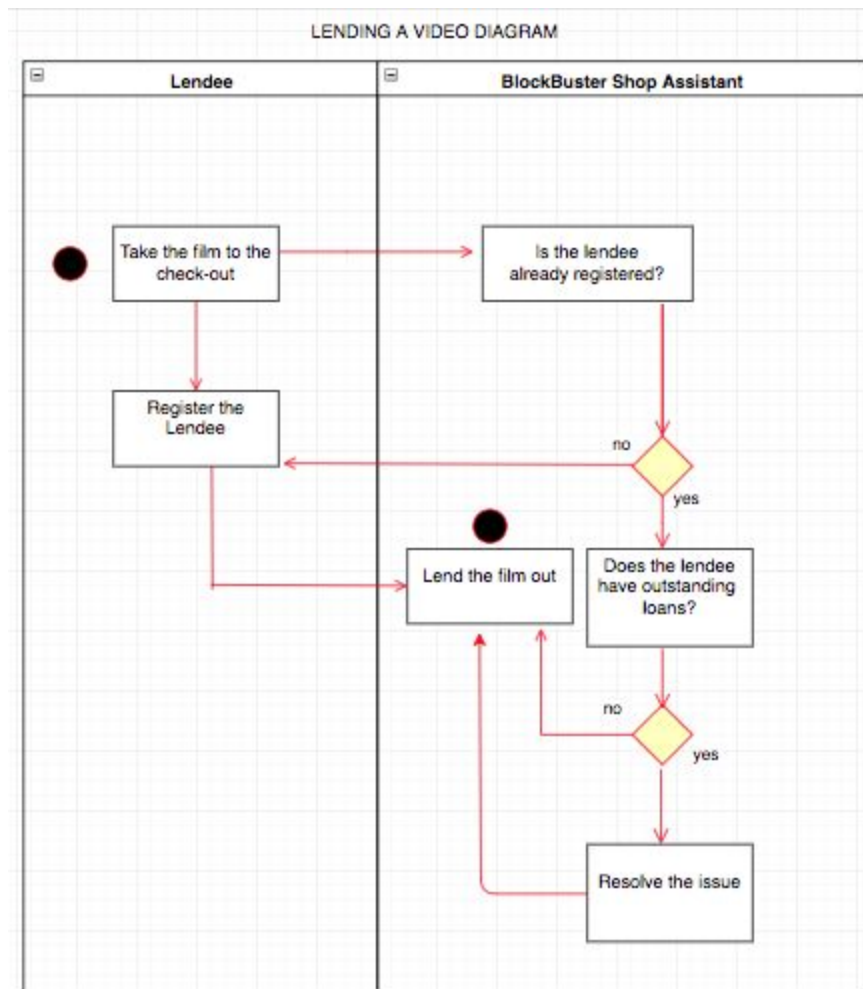
A Class Diagram



An Object Diagram



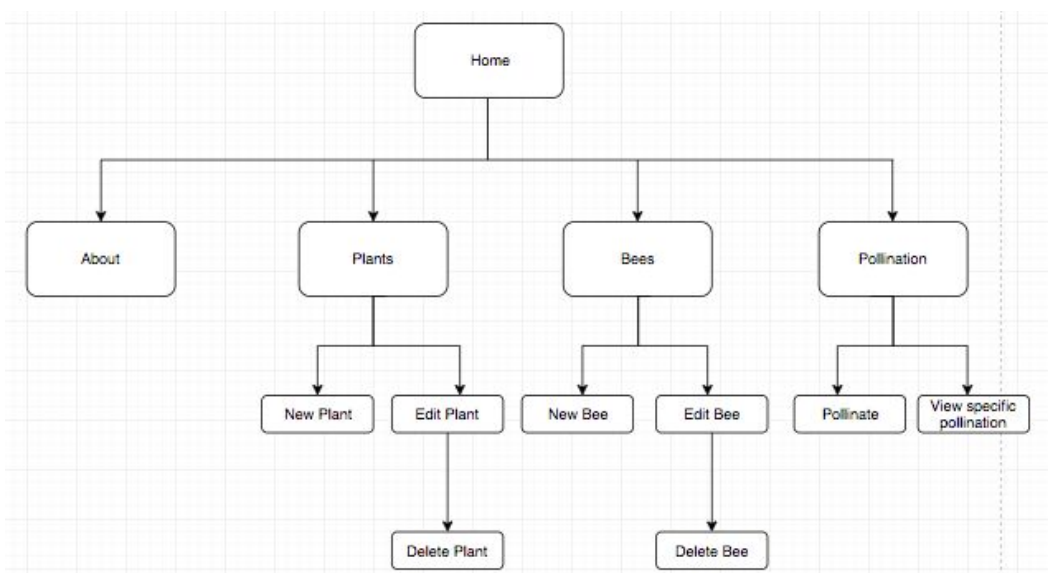
An Activity Diagram



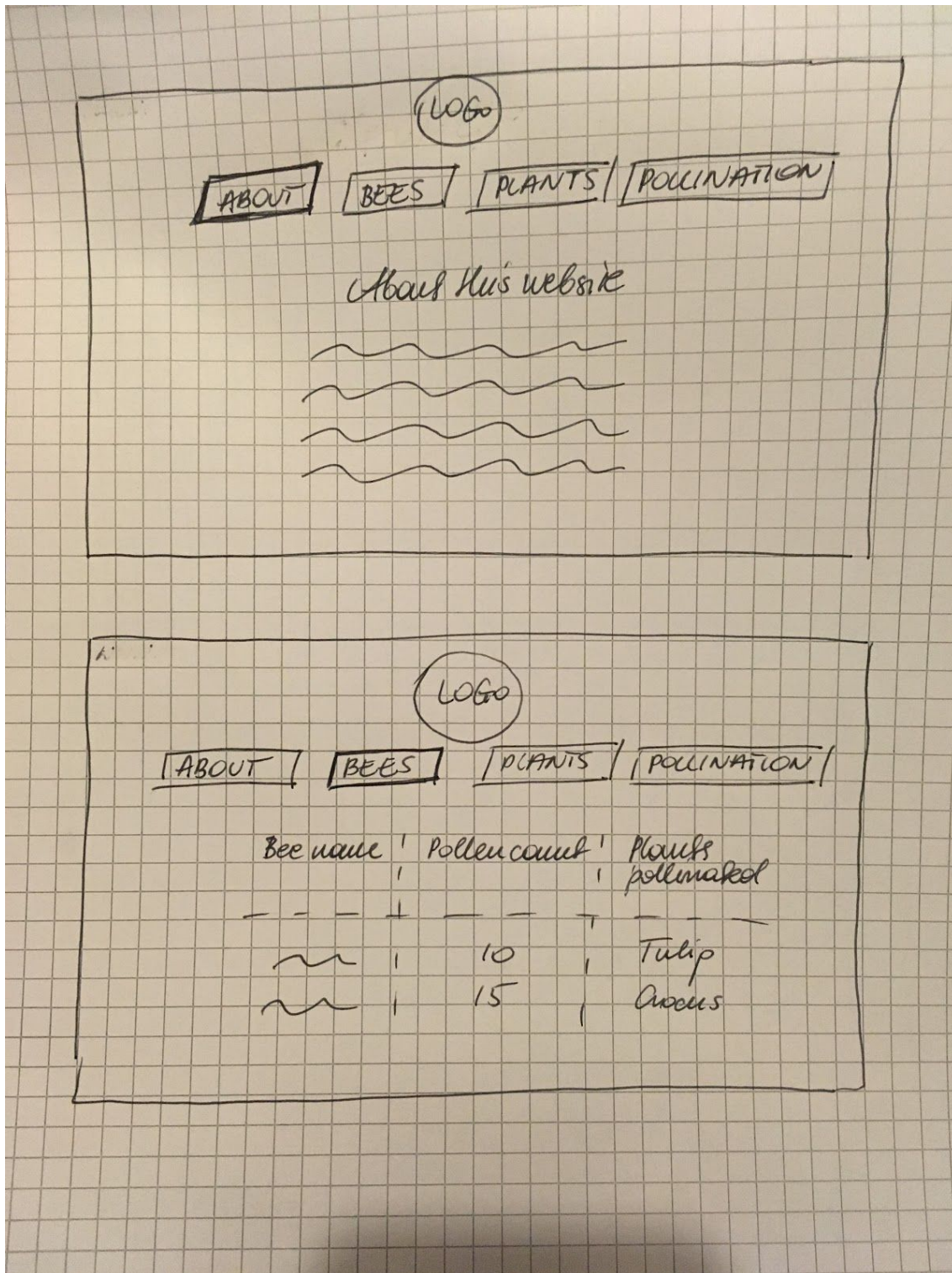
Implementations constraint

IMPLEMENTATIONS CONSTRAINT PLAN		
CONSTRAINT CATEGORY	IMPLEMENTATION CONSTRAINT	SOLUTION
HARDWARE AND SOFTWARE PLATFORMS	The development was carried out using a Mac and a Chrome browser, it was not made responsive when viewed on platforms different to the ones the development was made on due to time constraints. Lack of optimisation lead to limited usability and accessibility.	Ensure the product has responsive design & test on different platforms.
PERFORMANCE REQUIREMENTS	Response Time may be affected by slow internet connection and there might be display differences if viewed on mobile; potential cross-browser display issues, therefore limited usability and accessibility of the product.	Test the product's response time (using dev tools), & test on different platforms.
PERSISTENCE STORAGE AND TRANSACTIONS	No login system - data is not secured; the user would have to set up their own database to use the product, which limits accessibility.	This is a given constraint of the project. Reliable database system used (PostgreSQL) to persist data.
USABILITY	Product is not functional across different platforms due to time constraints.	Use responsible design & test the product on different platforms to improve usability.
BUDGET	No budget. Simple design and no proof-of-concept limits the product.	This is a given constraint of the project.
TIME LIMITATION	Project completed in 7 days, as part of Codeclan software development training.	This is a given constraint of the project. Additional functionality to be considered after the project deadline.

User Site Map



2 Wireframe Diagrams




Example of pseudocode used for a method

```
# check if hunger level and pollen together is less or equal 100
# if yes, return true
# if no, return false
def is_hungry_enough(pollen)
  if @hunger_level + pollen <= 100
    return true
  else
    return false
  end
end
```

Show user input being processed according to design requirements.

- The user inputting something into your program.
- The user input being saved or used in some way.



Home More Info Bees Plants Pollinate


Plant a new plant below

Name:

Select Season to Plant:

Pollen (max 100):

PLANT IT!



Home More Info Bees Plants Pollinate

You successfully created Tulip


Season to plant: *Spring*

Pollen: 50

POLLINATE IT

Show an interaction with data persistence.

- Data being inputted into a program.
- Confirmation of the data being saved.



Home More Info Bees Plants Pollinate


Create a new Bee below

Name:

Select Type:

How hungry? (max 100):

BECOME



Home More Info Bees Plants Pollinate

You successfully created Kirstin the Bee

Type: *Bumblebee*


How hungry is your new bee?

→ Kirstin is full

POLLINATE!

Show the correct output of results and feedback to user.

- The user requesting information or an action to be performed.
- The user request being processed correctly and demonstrated in the program.



Home More Info Bees Plants Pollinate

Pollination time

Select a bee:

Select a plant:

POLLINATE!

DELETE

Joanna flew over to Judas Tree

DELETE

Michael pollinated Elephant Ear

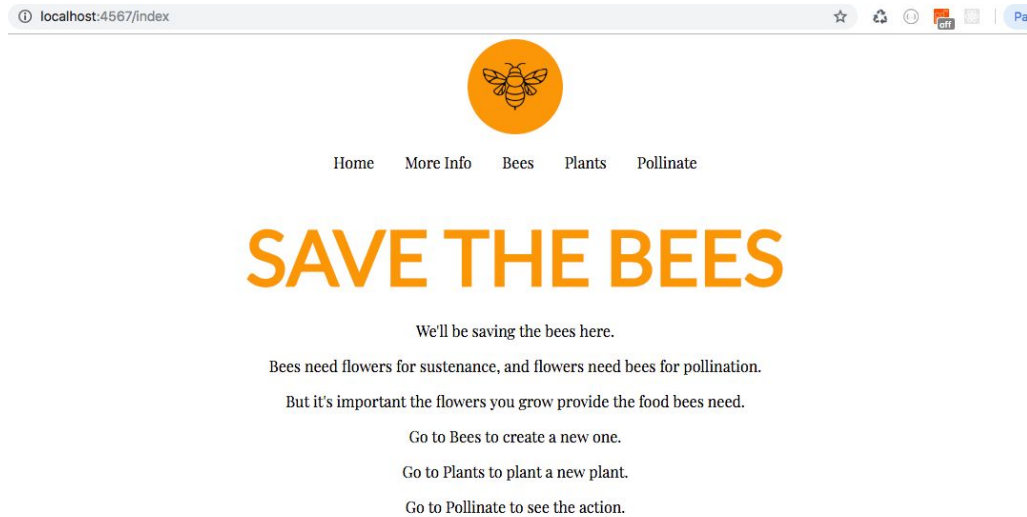
DELETE

Kirstin sat on Dahlia

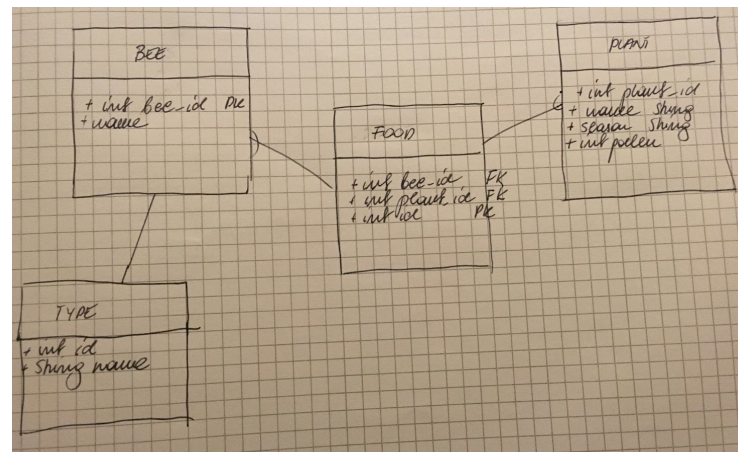
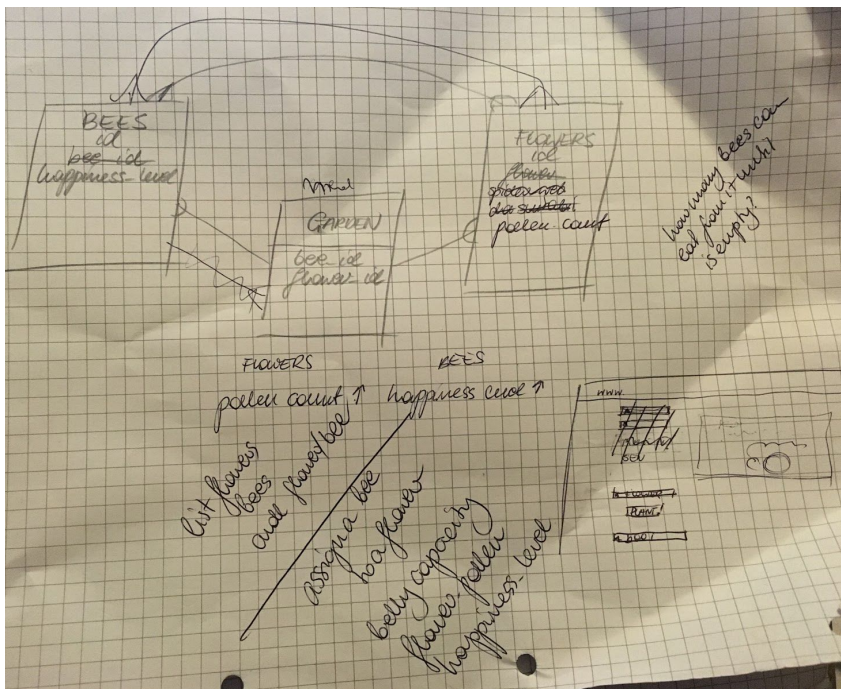
DELETE

Take a screenshots of one of your projects where you have worked alone and attach a GitHub link.

https://github.com/Beata-Ficek/RUBY-SINATRA_solo_project



Take screenshots or photos of your planning and the different stages of development to show changes.



WEEK 7.

Show an API being used within your program.

- The code that uses or implements API.
- The API being used by the program whilst running.

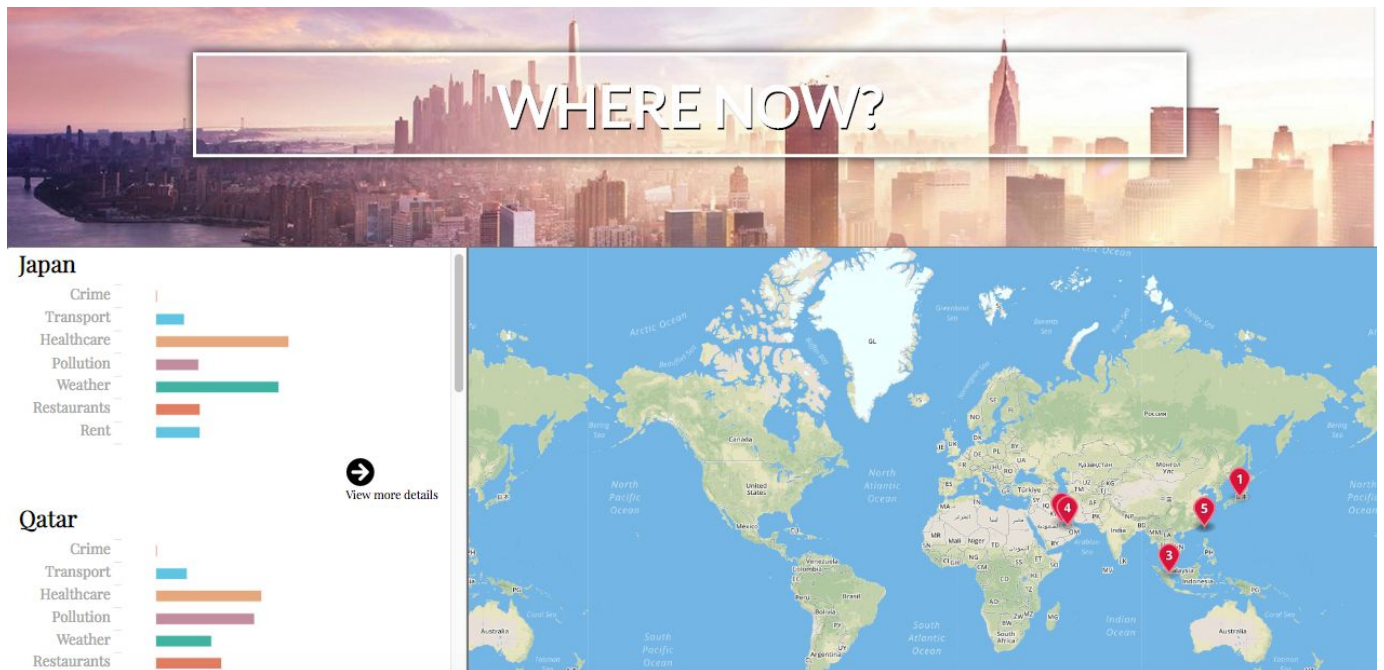
```
const Request = require('../helpers/request.js');
const PubSub = require('../helpers/pub_sub.js');

const Countries = function () {
  this.countryList = null;
  this.request = new Request("https://restcountries.eu/rest/v2/all");
};

Countries.prototype.bindEvents = function () {
  this.getData();
};

Countries.prototype.getData = function() {
  this.request.get()
    .then((countryList) => {
      PubSub.publish('Countries:countries-list-ready', countryList);
    })
    .catch(console.error);
};

module.exports = Countries;
```



Demonstrate testing in your program.

- Example of test code.
- The test code failing to pass.
- Example of test code once errors have been corrected.
- The test code passing.

```
task1.rb x cardGame.rb x card.rb x testing_ta
1 require("minitest/autorun")
2 require_relative("../card.rb")
3
4
5
6 class CardTest < Minitest::Test
7   def setup
8     @card1 = Card.new("ace", 0)
9     @card2 = Card.new("queen", 8)
10    @card3 = Card.new("king", 5)
11    @card4 = Card.new("joker", 2)
12    @game1 = CardGame.new
13  end
14
15  def test_check_for_ace_false
16    assert_equal(false, @game1.check_for_ace(@card2))
17  end
18
19  def test_highest_card_queen
20    assert_equal("queen", @game1.highest_card(@card1, @card2))
21  end
22
23  def test_highest_card_king
24    assert_equal("king", @game1.highest_card(@card1, @card3))
25  end
26
27  def test_cards_total_8
28    assert_equal("Cards' value totals to 8", CardGame.cards_total([@card1, @card2]))
29  end
30
31  def test_cards_total_7
32    assert_equal("Cards' value totals to 7", CardGame.cards_total([@card3, @card4]))
33  end
34
35 end
36
```

```
→ coding_exercise ruby specs/testing_task2.rb
Run options: --seed 1861

# Running:

EEEEEE

Finished in 0.001423s, 3513.7034 runs/s, 0.0000 assertions/s.

  1) Error:
CardTest#test_cards_total_7:
NameError: uninitialized constant CardTest::CardGame
Did you mean?  CardTest
               specs/testing_task2.rb:12:in `setup'

  2) Error:
CardTest#test_cards_total_8:
NameError: uninitialized constant CardTest::CardGame
Did you mean?  CardTest
               specs/testing_task2.rb:12:in `setup'

  3) Error:
CardTest#test_highest_card_king:
NameError: uninitialized constant CardTest::CardGame
Did you mean?  CardTest
               specs/testing_task2.rb:12:in `setup'

  4) Error:
CardTest#test_check_for_ace_false:
NameError: uninitialized constant CardTest::CardGame
Did you mean?  CardTest
               specs/testing_task2.rb:12:in `setup'

  5) Error:
CardTest#test_highest_card_queen:
NameError: uninitialized constant CardTest::CardGame
Did you mean?  CardTest
               specs/testing_task2.rb:12:in `setup'

5 runs, 0 assertions, 0 failures, 5 errors, 0 skips
→ coding_exercise
```

Test Code Failing.


```
testing_task2.rb — ~/codeclan_work/PDA_Evidence/coding_exercise
task1.rb x cardGame.rb x card.rb x testing_ta
1 require("minitest/autorun")
2 require_relative("../card.rb")
3 require_relative("../cardGame.rb")
4
5
6
7 class CardTest < Minitest::Test
8   def setup
9     @card1 = Card.new("ace", 0)
10    @card2 = Card.new("queen", 8)
11    @card3 = Card.new("king", 5)
12    @card4 = Card.new("joker", 2)
13    @game1 = CardGame.new
14  end
15
16  def test_check_for_ace__false
17    assert_equal(false, @game1.check_for_ace(@card2))
18  end
19
20  def test_highest_card__queen
21    assert_equal("queen", @game1.highest_card(@card1, @card2))
22  end
23
24  def test_highest_card__king
25    assert_equal("king", @game1.highest_card(@card1, @card3))
26  end
27
28  def test_cards_total__8
29    assert_equal("Cards' value totals to 8", CardGame.cards_total([@card1, @card2]))
30  end
31
32  def test_cards_total__7
33    assert_equal("Cards' value totals to 7", CardGame.cards_total([@card3, @card4]))
34  end
35
36 end

coding_exercise — bea@MacUsers-iMac — ..ding_exercis...
[→ coding_exercise ruby specs/testing_task2.rb
Run options: --seed 37481

# Running:

.....

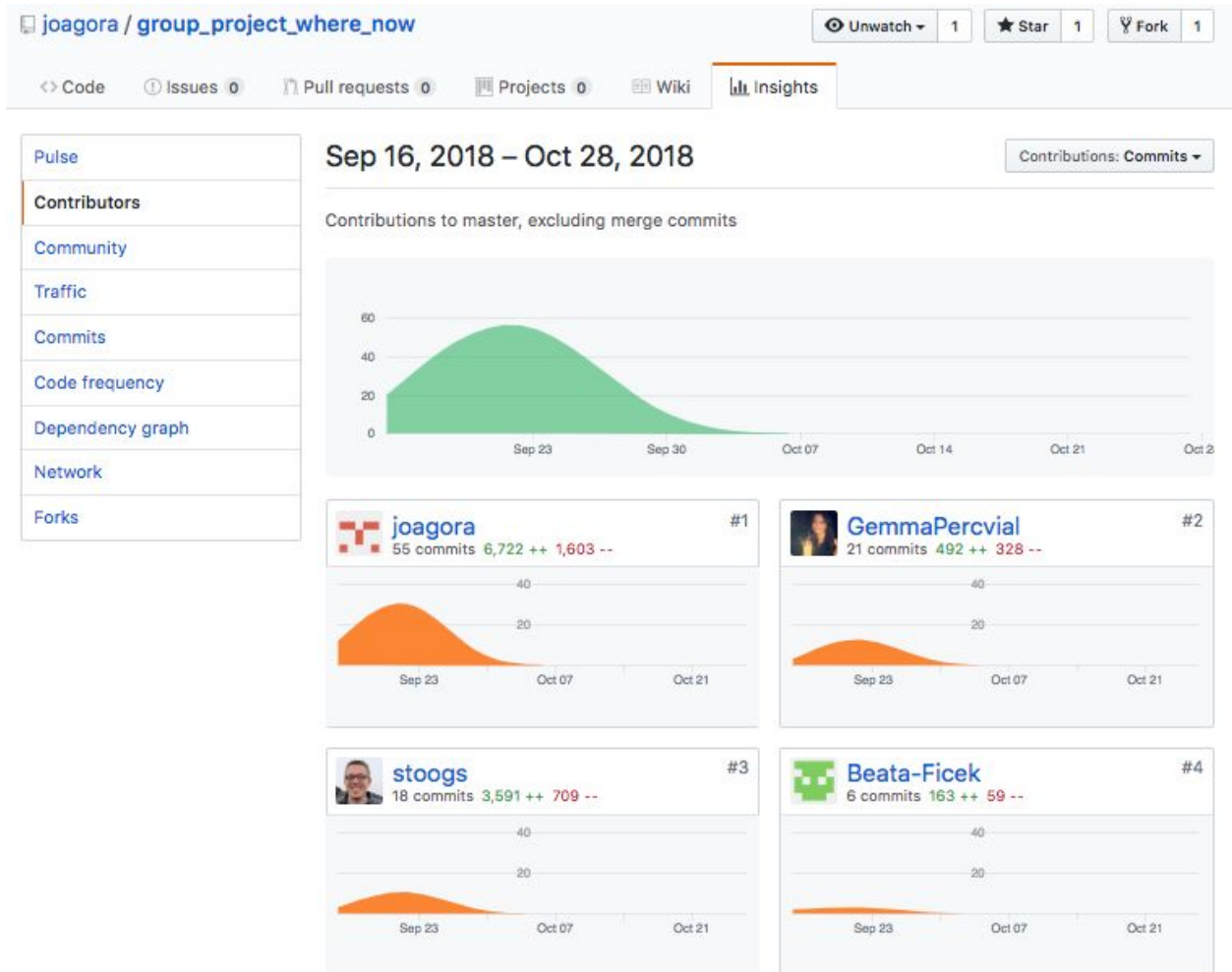
Finished in 0.001476s, 3387.5339 runs/s, 3387.5339 assertions
/s.

5 runs, 5 assertions, 0 failures, 0 errors, 0 skips
→ coding_exercise █
```

Test Code passing.

WEEK 9.

Screenshot of the contributor's page on GitHub from the group project to show the team you worked with.



Screenshot of the project brief from your group project.

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

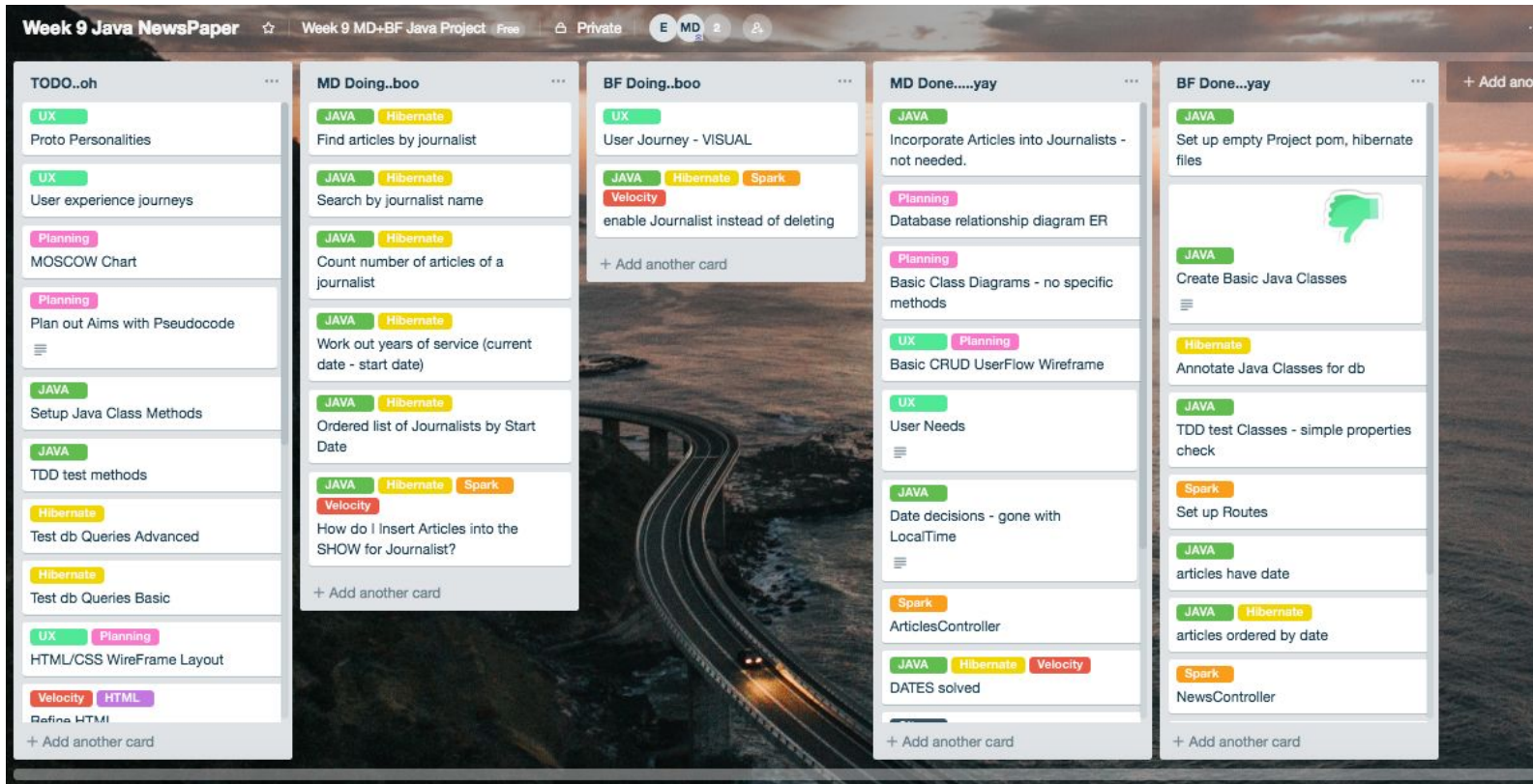
Example Extensions

- 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.

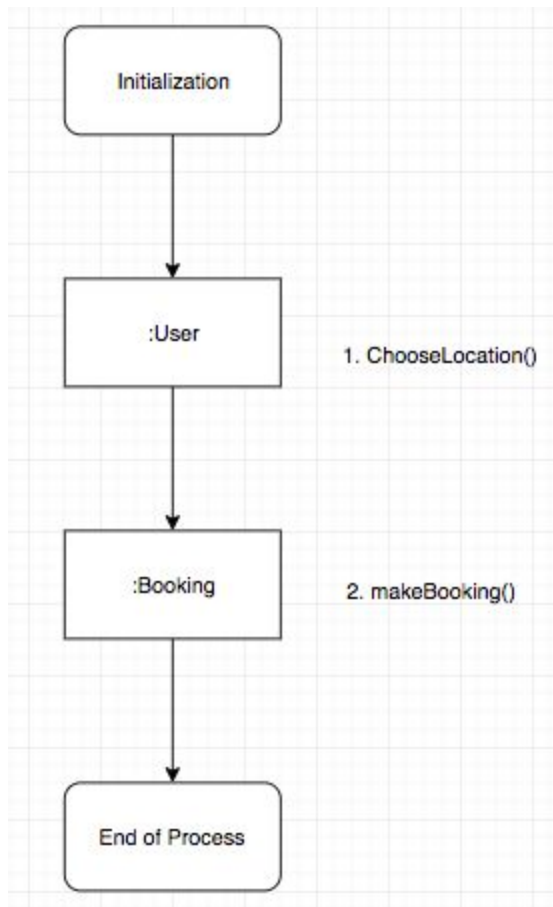
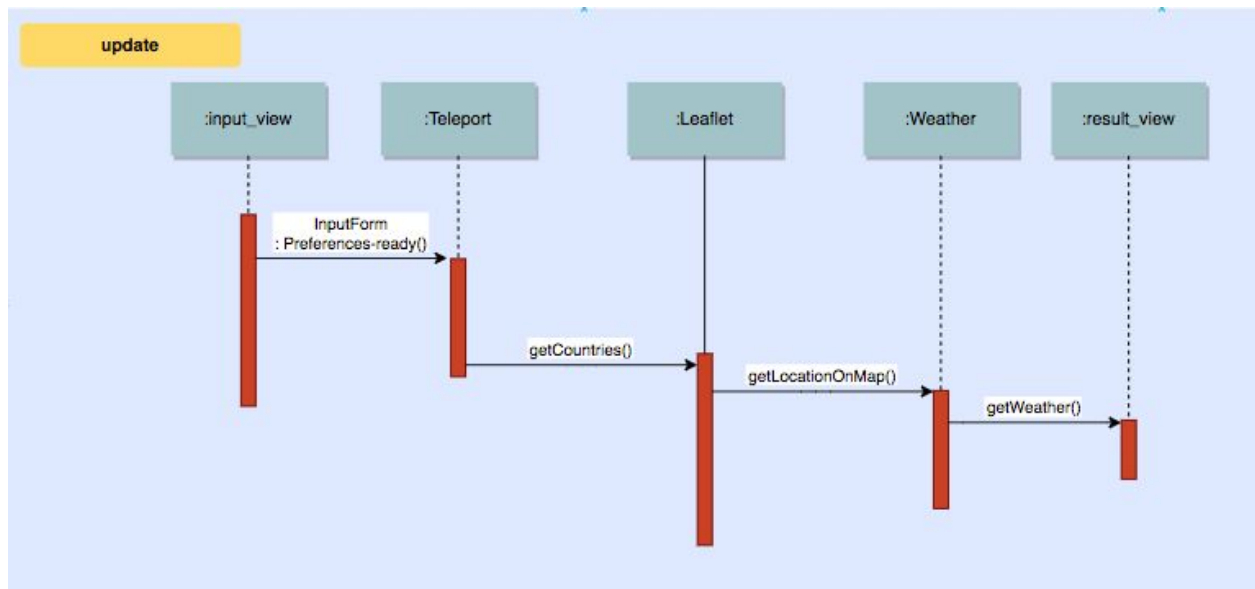
Screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.



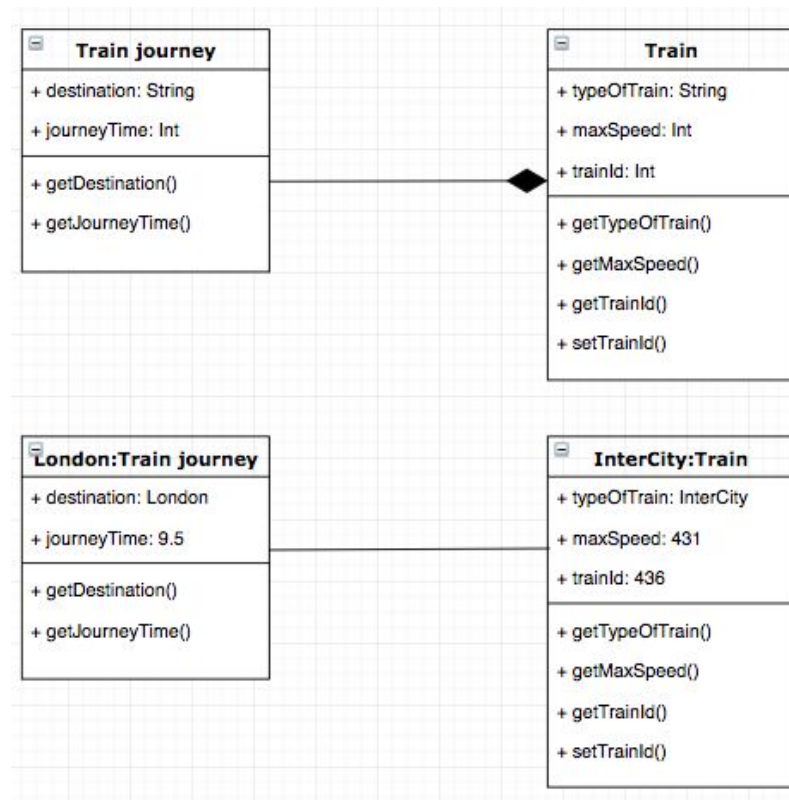
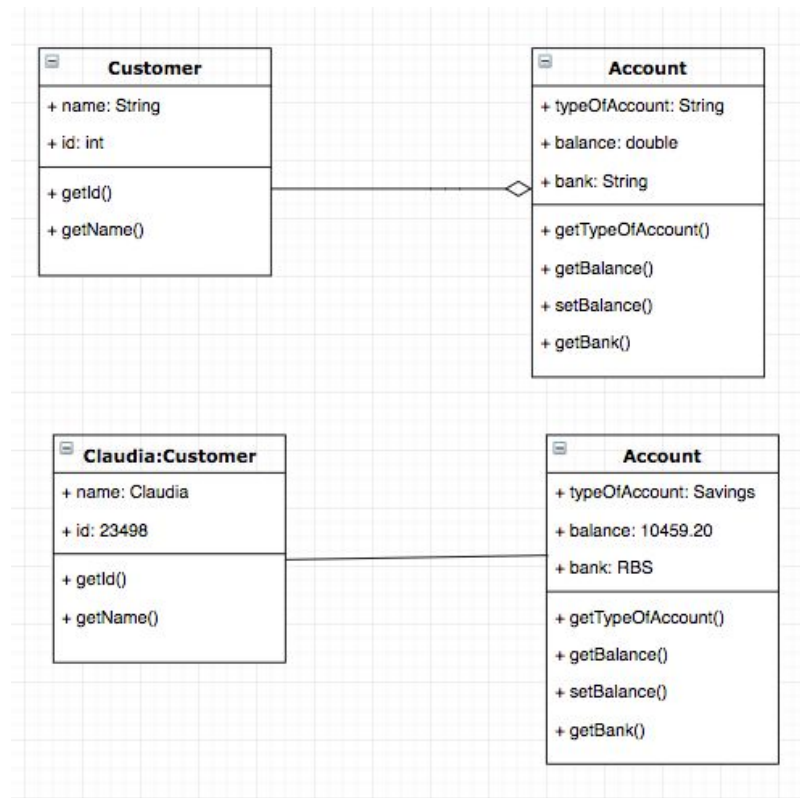
Write an acceptance criteria and test plan.

ACCEPTANCE CRITERIA	EXPECTED OUTCOME	PASS/ FAIL
User can enter a new article, edit it and delete it.	Provide a page containing all articles that can be edited or deleted. There is an option to add a new article.	PASS
User can add, edit or delete a journalist.	Provide a page containing all Journalists working for the company - a journalist can be edited or deleted. There is an option to add a new Journalist.	PASS
User can search the articles by title of the article.	Search bar that would match user's entry with the words in the title and provide the user with a list of articles displayed on the front page.	PASS
User can filter articles by a particular journalist	Dropdown containing a list of journalists. When a journalist gets chosen from the dropdown their articles are displayed on the front page.	PASS

Produce two system interaction diagrams (sequence and/or collaboration diagrams)



Produce two Object Diagrams



Produce a bug tracking report.

Bug / Error	Solution	Date
When filtering countries by preferences that the user put in the same results render despite change in preferences.	Investigated API. Filtered countries by "quality_of_life" to ensure that the countries have all the data necessary for the filtering process.	20/09/18
Unable to run the program.	Resolved CORS issues.	19/09/18
Title of the page is not rendered when in the result view.	Change the position of the title using CSS.	18/09/18
Countries from two different API's not matching.	Match the countries using country code, instead of country name.	17/09/18
Page elements not fitting into Main Container on the front page.	Adjust CSS code using Flexbox.	16/09/18

WEEK 12.

The use of polymorphism in a program and what it's doing.

```
public class Piano extends Instrument implements IPlay, ISell {
    private String manufacturer;
    private int buyPrice;
    private int sellPrice;

    public Piano(String manufacturer, String colour, int buyPrice, int sellPrice) {
        super(colour, "Keyboard");
        this.manufacturer = manufacturer;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }

    public String getManufacturer() {
        return this.manufacturer;
    }

    public int getBuyPrice() {
        return this.buyPrice;
    }

    public int getSellPrice() {
        return this.sellPrice;
    }

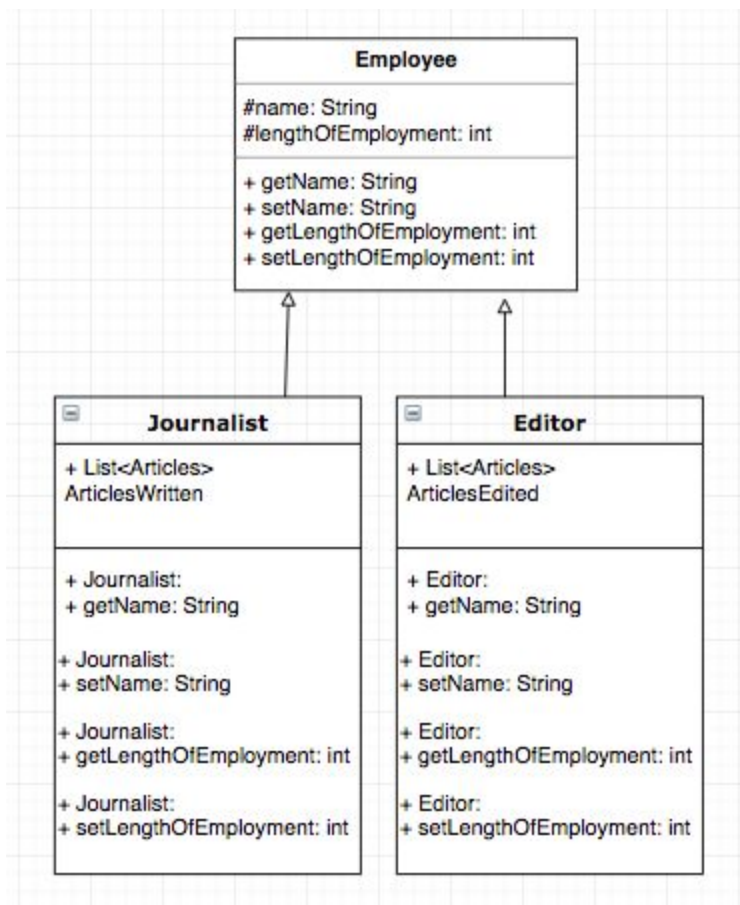
    public int calculateMarkup() {
        return this.sellPrice - this.buyPrice;
    }

    public String play() {
        return "Plink Plonk";
    }
}
```

Polymorphism means that an object can take many forms - this is achieved through using super classes or interfaces. In this example Piano uses the IPlay and ISell interfaces (and extends Instrument). This means that the Instrument can be a Guitar and it can also be sellable.

```
public interface IPlay {
    String play();
}
```


An Inheritance Diagram.



The use of encapsulation in a program and what it's doing.

```
@Entity
@Table(name = "categories")
public class Category {

    private int id;
    private String name;
    private List<Article> articles;

    public Category(String name){
        this.name = name;
        this.articles = new ArrayList<Article>();
    }

    public Category(){

    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

Encapsulation is hiding data inside a class to prevent unauthorized parties' direct access to it. In the above example id is set to "private" so it's only accessible to the relevant parties. However, below that id is used in "public" methods getId and setId, so that id can be accessed when needed.

Use of Inheritance in a program.

- 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.

```
public abstract class Staff {

    private int id;
    private String name;
    private LocalDate startDate;
    private int yearsOfService;
    // private List<Article> articles;

    public Staff(){

    }

    public Staff(String name, LocalDate date, int yearsOfService){
        this.name = name;
        this.startDate = date;
        this.yearsOfService = yearsOfService;
        // this.articles = new ArrayList<Article>();
    }
}
```

Staff is an Abstract Superclass

```

@Entity
@Table(name= "journalists")
public class Journalist extends Staff {

    private List<Article> articles;
    public Journalist(){

    }

    public Journalist(String name, LocalDate date, int yearsOfService) {
        super(name, date, yearsOfService);
        this.articles = new ArrayList<>();
    }
}

```

Journalist Class
inherits from Staff

Journalist Class having name, startDate and yearsOfService properties from Staff.

```

// CREATE = Save new details to DB
post ( path: "/journalists", (req, res)->{
    String name = req.queryParams("name");
    String startDateString = req.queryParams("startDate");
    int yearsOfService = Integer.parseInt(req.queryParams("yearsOfService"));
}

```

Able to getName of a Journalist.

```

@Test
public void hasName() {
    assertEquals( expected: "Jim", journalist1.getName());
}

```

Two algorithms you have written. Short statement on why you have chosen to use those algorithms.

```
def how_hungry()  
  if @hunger_level <= 40  
    return "is very hungry"  
  elsif @hunger_level.between?(41, 70)  
    return "is okay"  
  elsif @hunger_level >= 71  
    return "is full"  
  end  
end
```

This algorithm uses an if/elsif statement which checks the @hunger_level and returns a message depending on the hunger level.

```
def is_hungry_enough(pollen)  
  if @hunger_level + pollen <= 100  
    return true  
  else  
    return false  
  end  
end
```

This algorithm checks if @hunger_level and pollen together is equal or less than 100. If it is less, the algorithm returns true; if not, it returns false.

