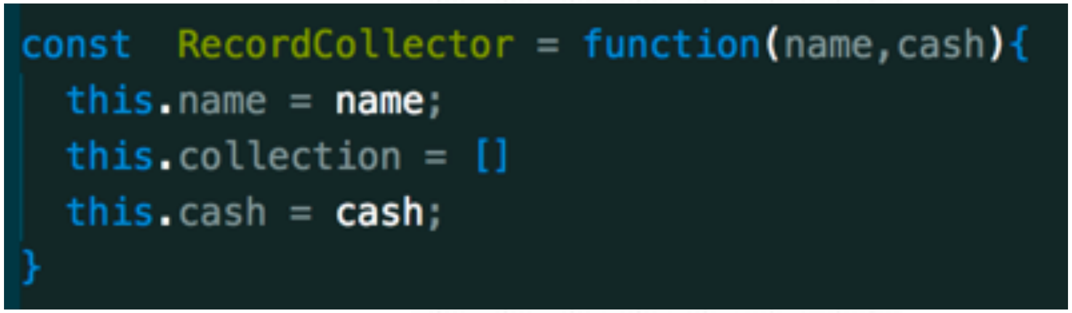
PDA: Software Development
Level 8
Student Evidence Checklist

Full name	**Christopher Murphy**
Cohort	**G4**

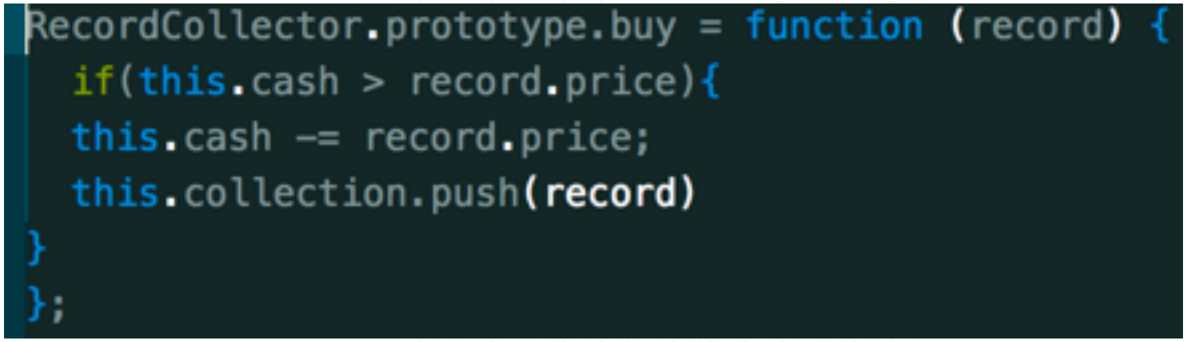
The evidence required can be taken from your assignments, homework that you have completed on your own or by creating a specific example for the PDA.

<table>
<tr><th></th><th>Unit</th><th>Ref.</th><th>Evidence</th><th>Done</th></tr>
<tr><td rowspan="3"><b>Week 2</b></td><td>I &amp; T</td><td>I.T 5</td><td>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</td><td></td></tr>
<tr><td></td><td></td><td>



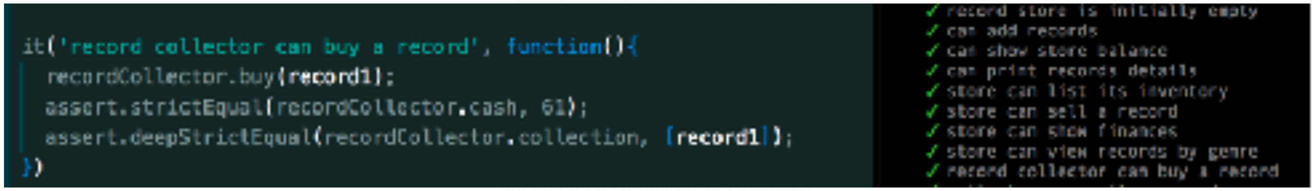
```
const  RecordCollector = function(name,cash){
  this.name = name;
  this.collection = []
  this.cash = cash;
}
```

</td><td></td></tr>
<tr><td></td><td></td><td>



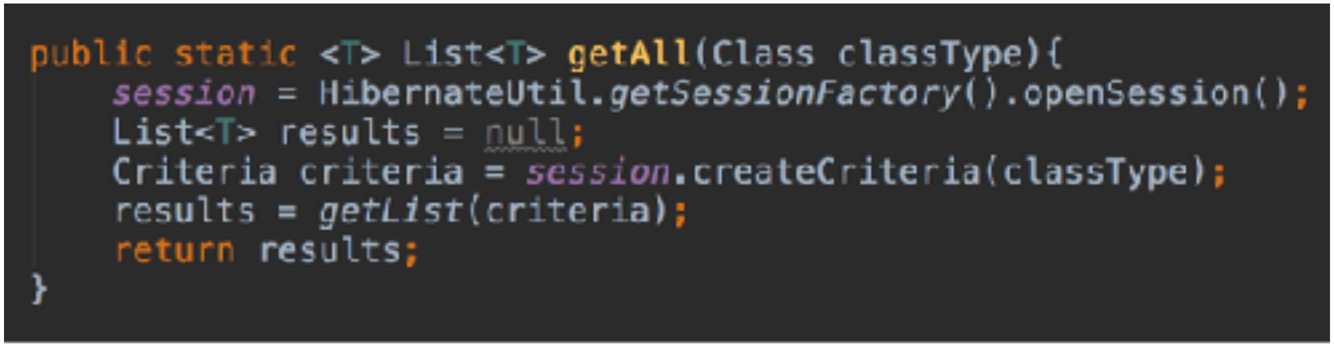
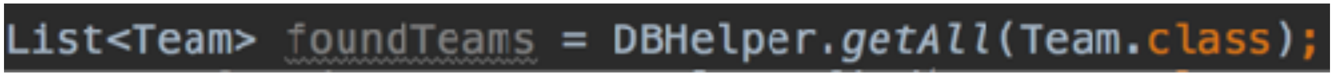
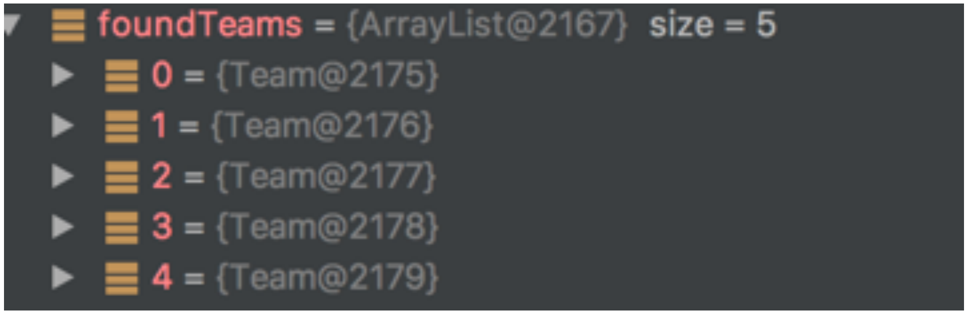
```
RecordCollector.prototype.buy = function (record) {
  if(this.cash > record.price){
  this.cash -= record.price;
  this.collection.push(record)
}
};
```

</td><td></td></tr>
</table>

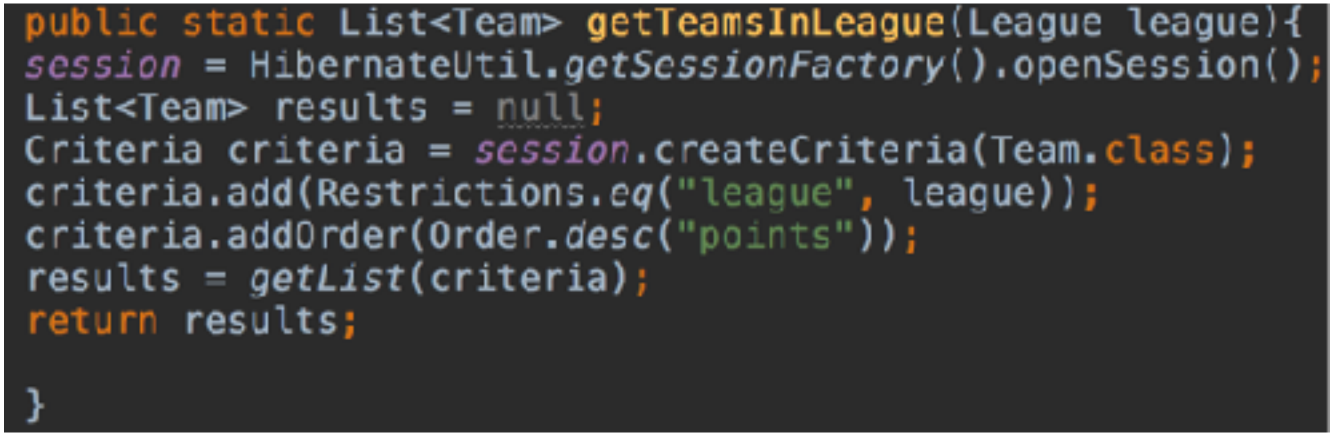
			
			`it('record collector can buy a record', function(){`
			`  recordCollector.buy(record1);`
			`  assert.strictEqual(recordCollector.cash, 61);`
			`  assert.deepStrictEqual(recordCollector.collection, [record1]);`
			`})`
			`✓ record store is initially empty`
			`✓ can add records`
			`✓ can show store balance`
			`✓ can print records details`
			`✓ store can list its inventory`
			`✓ store can sell a record`
			`✓ store can show finances`
			`✓ store can view records by genre`
			`✓ record collector can buy a record`	

	I & T	I.T 6		
	I & T		Static and Dynamic testing task A	

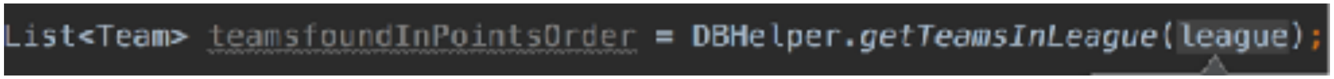
	Unit	**Ref.**	**Evidence**	**Done**
	I & T	I.T 3	Demonstrate searching data in a program. Take screenshots of: *Function that searches data *The result of the function running	

Week 3			
			`public static <T> List<T> getAll(Class classType){`
			`    session = HibernateUtil.getSessionFactory().openSession();`
			`    List<T> results = null;`
			`    Criteria criteria = session.createCriteria(classType);`
			`    results = getList(criteria);`
			`    return results;`
			`}`
			
			`List<Team> foundTeams = DBHelper.getAll(Team.class);`
			
			`foundTeams = {ArrayList@2167}  size = 5`
			`  0 = {Team@2175}`
			`  1 = {Team@2176}`
			`  2 = {Team@2177}`
			`  3 = {Team@2178}`
			`  4 = {Team@2179}`	

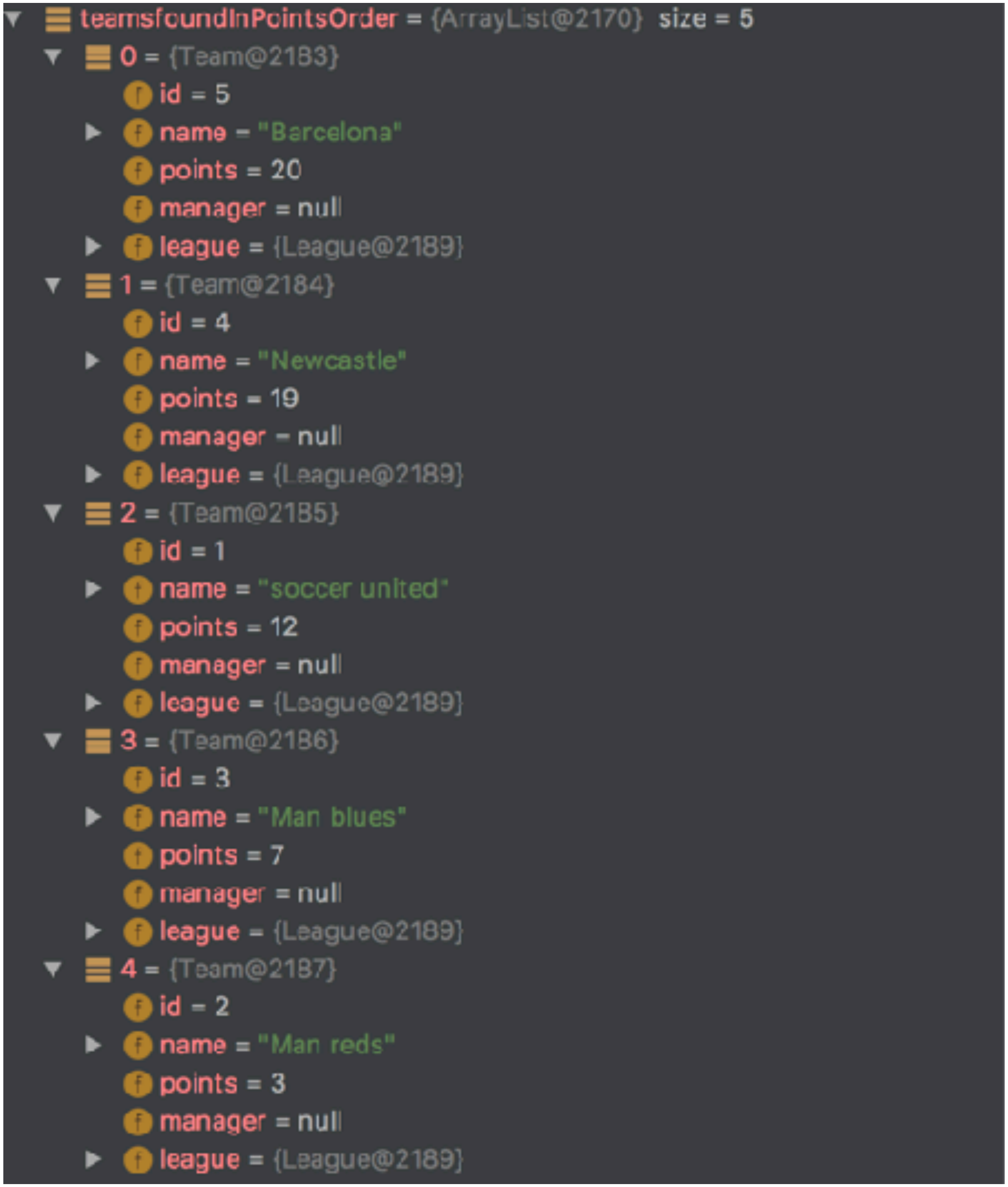
	I & T	I.T 4	Demonstrate sorting data in a program. Take screenshots of: *Function that sorts data *The result of the function running	



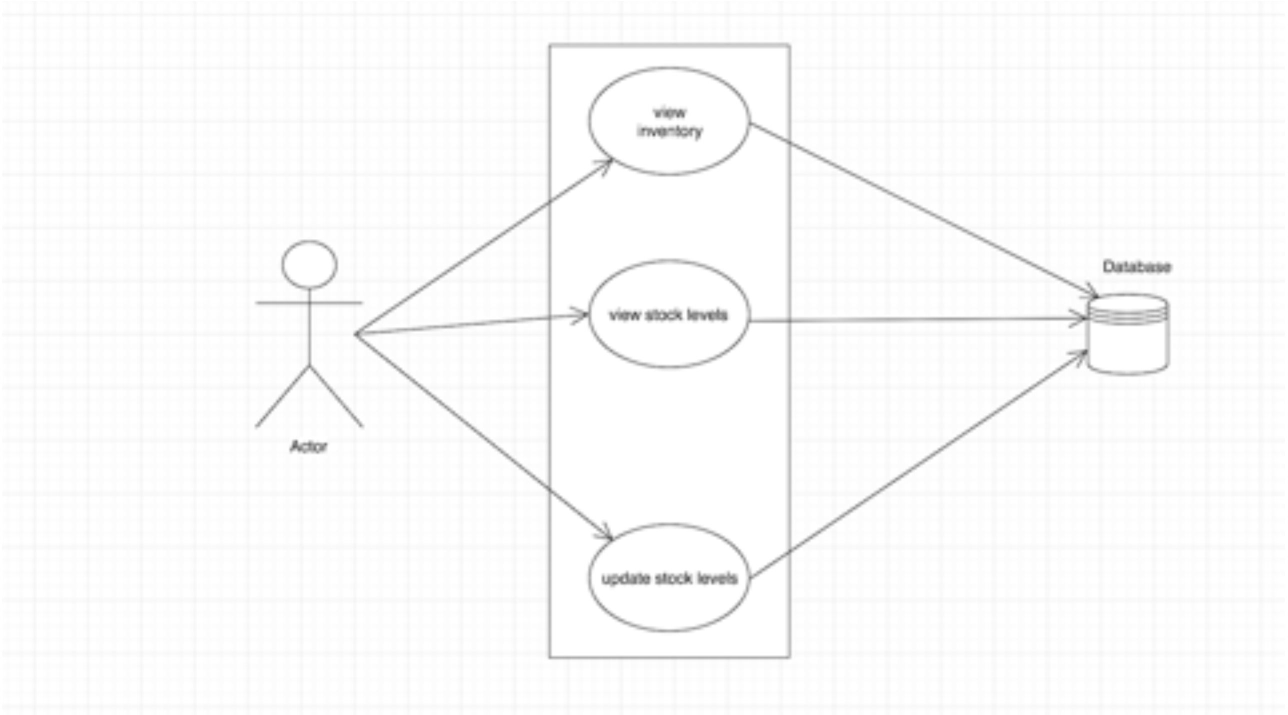
```
public static List<Team> getTeamsInLeague(League league){
session = HibernateUtil.getSessionFactory().openSession();
List<Team> results = null;
Criteria criteria = session.createCriteria(Team.class);
criteria.add(Restrictions.eq("league", league));
criteria.addOrder(Order.desc("points"));
results = getList(criteria);
return results;

}
```

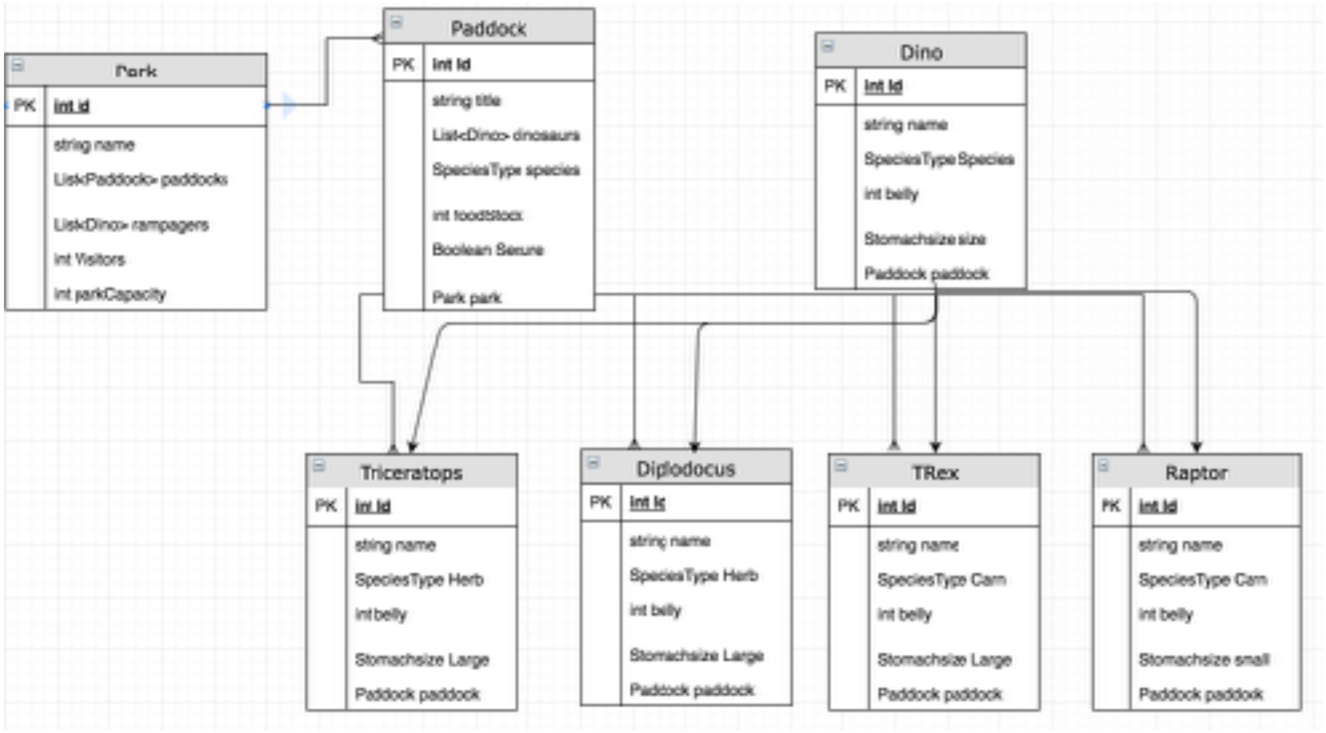
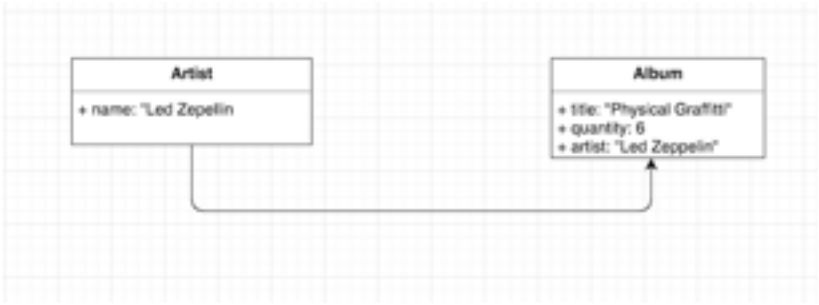


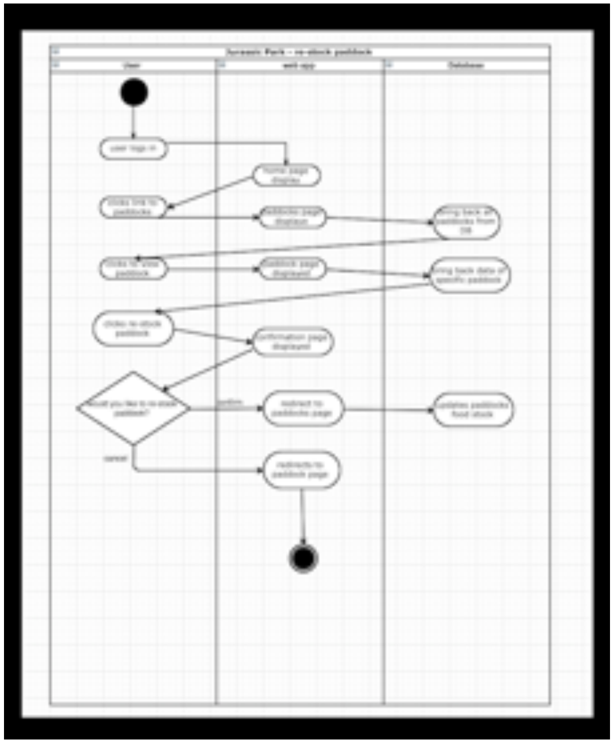
```
List<Team> teamsfoundInPointsOrder = DBHelper.getTeamsInLeague(league);
```

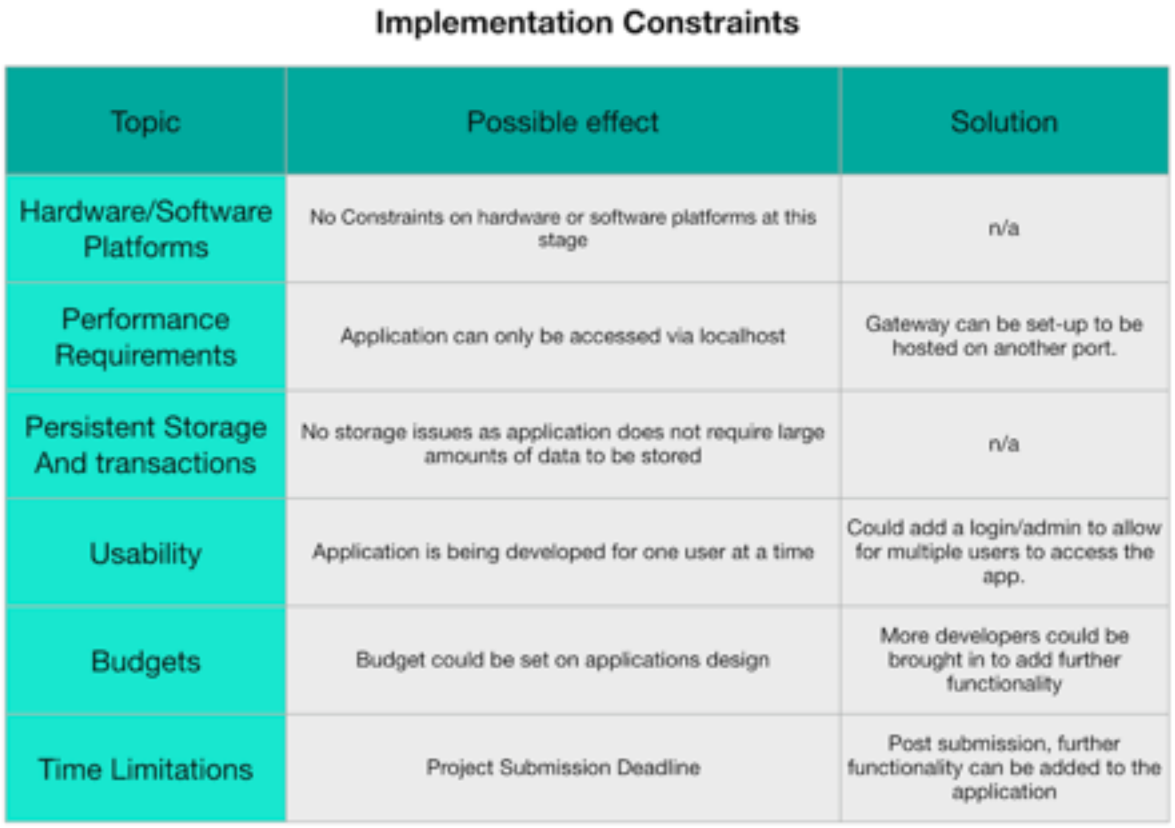
				

```
▼ teamsfoundInPointsOrder = {ArrayList@2170}  size = 5
    ▼ 0 = {Team@2183}
          id = 5
        ▶ name = "Barcelona"
          points = 20
          manager = null
        ▶ league = {League@2189}
    ▼ 1 = {Team@2184}
          id = 4
        ▶ name = "Newcastle"
          points = 19
          manager = null
        ▶ league = {League@2189}
    ▼ 2 = {Team@2185}
          id = 1
        ▶ name = "soccer united"
          points = 12
          manager = null
        ▶ league = {League@2189}
    ▼ 3 = {Team@2186}
          id = 3
        ▶ name = "Man blues"
          points = 7
          manager = null
        ▶ league = {League@2189}
    ▼ 4 = {Team@2187}
          id = 2
        ▶ name = "Man reds"
          points = 3
          manager = null
        ▶ league = {League@2189}
```

	Unit	Ref.	Evidence	Done
	A & D	A.D 1	A Use Case Diagram	
			 view inventory view stock levels update stock levels Actor Database	
	A & D	A.D 2	A Class diagram.	

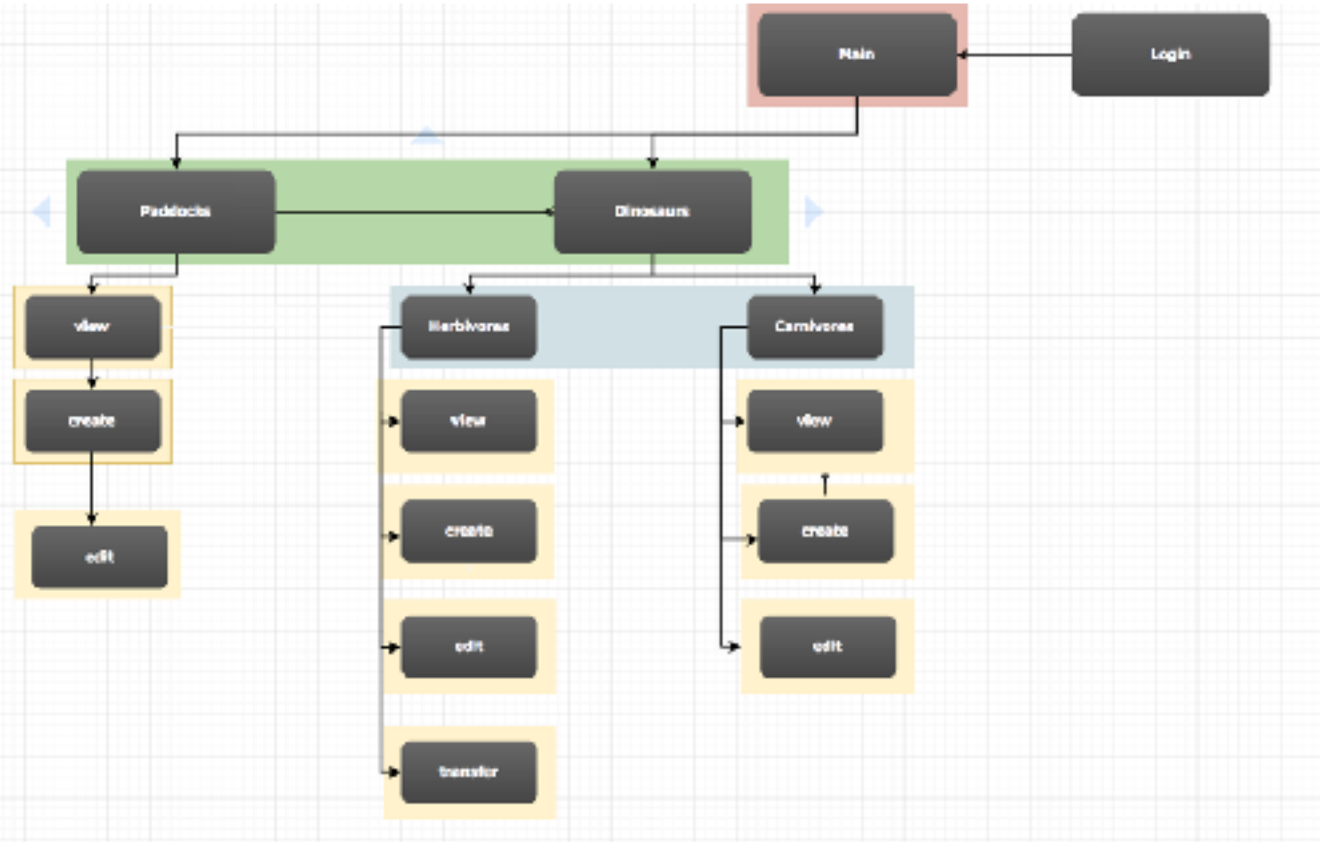
		 Park PK int id string name List<Paddock> paddocks List<Dino> rampagers int Visitors int parkCapacity Paddock PK int Id string title List<Dino> dinosaurs SpeciesType species int foodStock Boolean Secure Park park Dino PK int Id string name SpeciesType Species int belly Stomachsize size Paddock paddock Triceratops PK int Id sting name SpeciesType Herb int belly Stomachsize Large Paddock paddock Diplodocus PK int Id string name SpeciesType Herb int belly Stomachsize Large Paddock paddock TRex PK int Id string name SpeciesType Carn int belly Stomachsize Large Paddock paddock Raptor PK int Id string name SpeciesType Carn int belly Stomachsize small Paddock paddock	
A & D	A.D 3	An Object diagram.	
		 Artist + name: "Led Zepellin Album + title: "Physical Graffiti" + quantity: 6 + artist: "Led Zeppelin"	
A & D	A.D 4	An Activity Diagram	

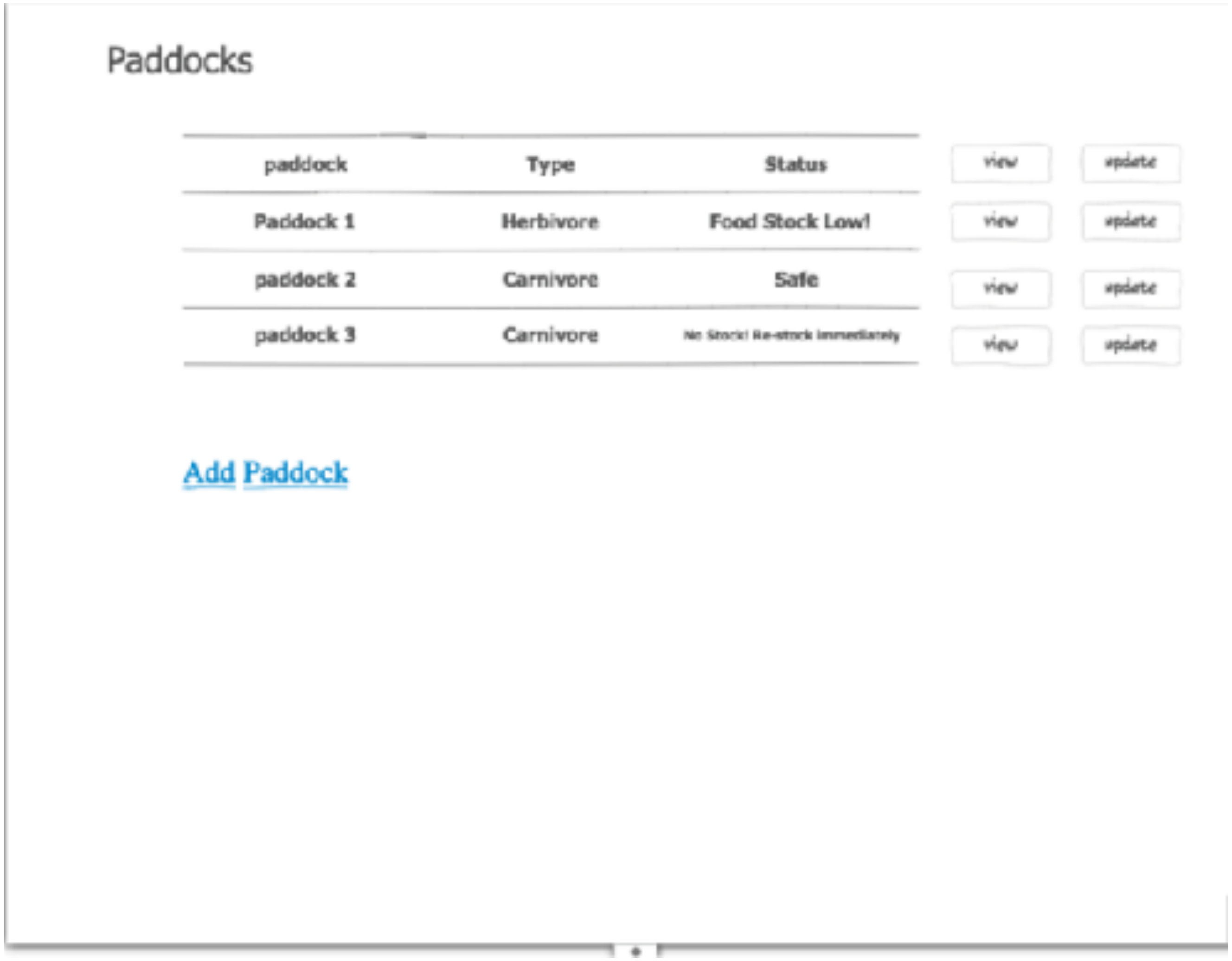
		 Jurassic Park – re-stock paddock User web app Database user logs in home page display clicks link to paddocks paddocks page displays bring back all paddocks from DB clicks to view paddock paddock page displayed bring back data of specific paddock clicks re-stock paddock confirmation page displayed would you like to re-stock paddock? confirm redirect to paddocks page updates paddocks food stock cancel redirects to paddock page	
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	

<table>
<tr><td></td><td></td><td>

Implementation Constraints

<table>
<tr><th>Topic</th><th>Possible effect</th><th>Solution</th></tr>
<tr><td>Hardware/Software Platforms</td><td>No Constraints on hardware or software platforms at this stage</td><td>n/a</td></tr>
<tr><td>Performance Requirements</td><td>Application can only be accessed via localhost</td><td>Gateway can be set-up to be hosted on another port.</td></tr>
<tr><td>Persistent Storage And transactions</td><td>No storage issues as application does not require large amounts of data to be stored</td><td>n/a</td></tr>
<tr><td>Usability</td><td>Application is being developed for one user at a time</td><td>Could add a login/admin to allow for multiple users to access the app.</td></tr>
<tr><td>Budgets</td><td>Budget could be set on applications design</td><td>More developers could be brought in to add further functionality</td></tr>
<tr><td>Time Limitations</td><td>Project Submission Deadline</td><td>Post submission, further functionality can be added to the application</td></tr>
</table>
</td><td></td></tr>
<tr><td>P</td><td>P 5</td><td>Create a user sitemap.</td><td></td></tr>
</table>

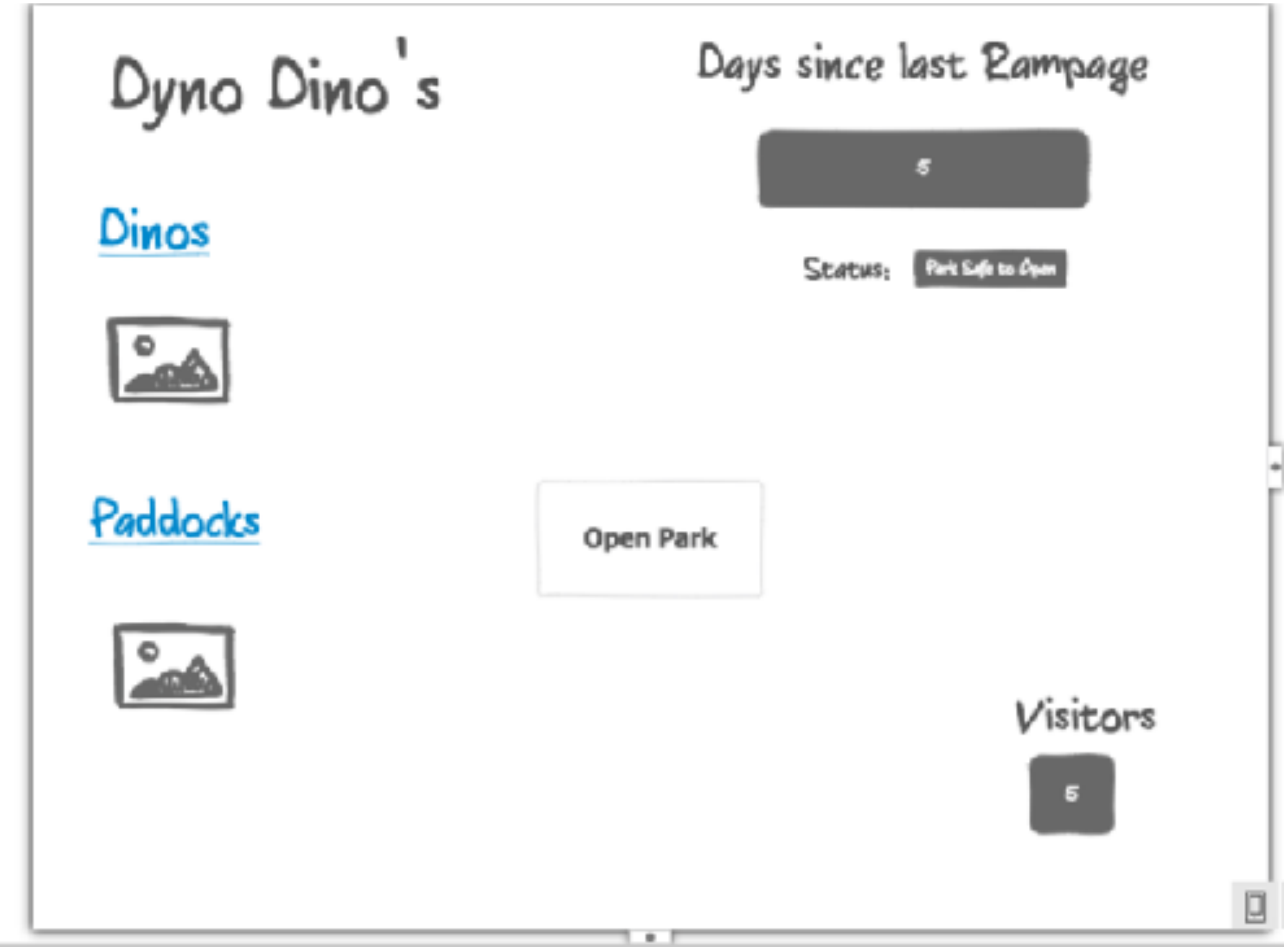
			
P	P 6	Produce two wireframe designs.	

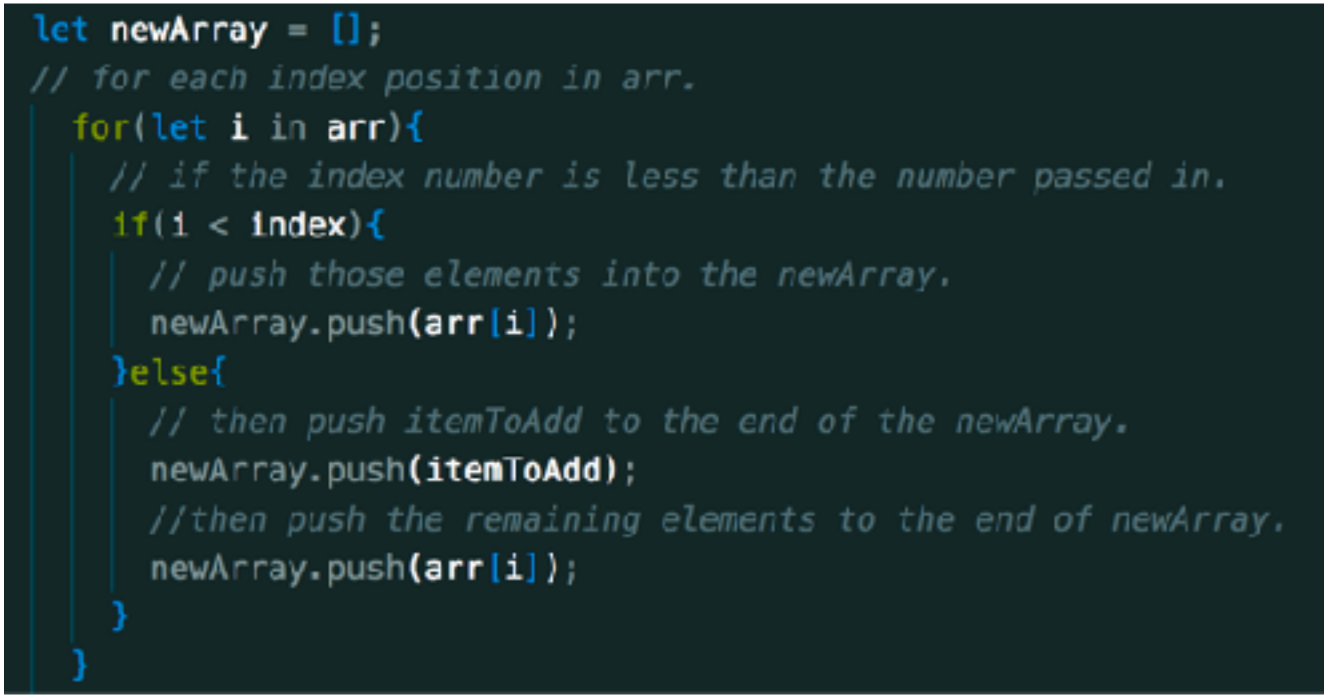
Week 5				

Paddocks

paddock	Type	Status	view	update
Paddock 1	Herbivore	Food Stock Low!	view	update
paddock 2	Carnivore	Safe	view	update
paddock 3	Carnivore	No Stock! Re-stock immediately	view	update

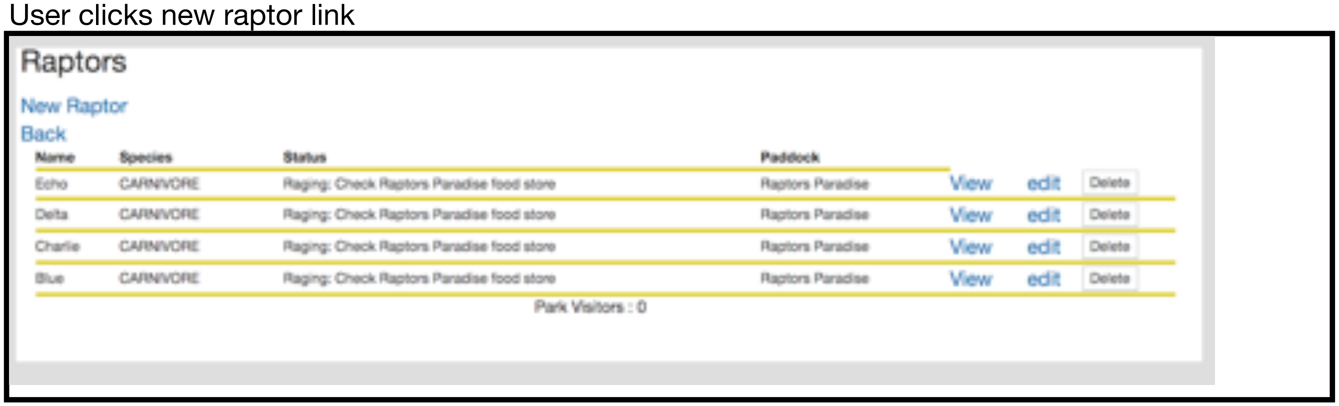
Add Paddock

		 Dyno Dino's Days since last Rampage 5 Dinos Status: Park Safe to Open Paddocks Open Park Visitors 5	
P	P 10	Take a screenshot of an example of pseudocode for a function.	

			
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	

```
let newArray = [];
// for each index position in arr.
  for(let i in arr){
    // if the index number is less than the number passed in.
    if(i < index){
      // push those elements into the newArray.
      newArray.push(arr[i]);
    }else{
      // then push itemToAdd to the end of the newArray.
      newArray.push(itemToAdd);
      //then push the remaining elements to the end of newArray.
      newArray.push(arr[i]);
    }
  }
```

User clicks new raptor link



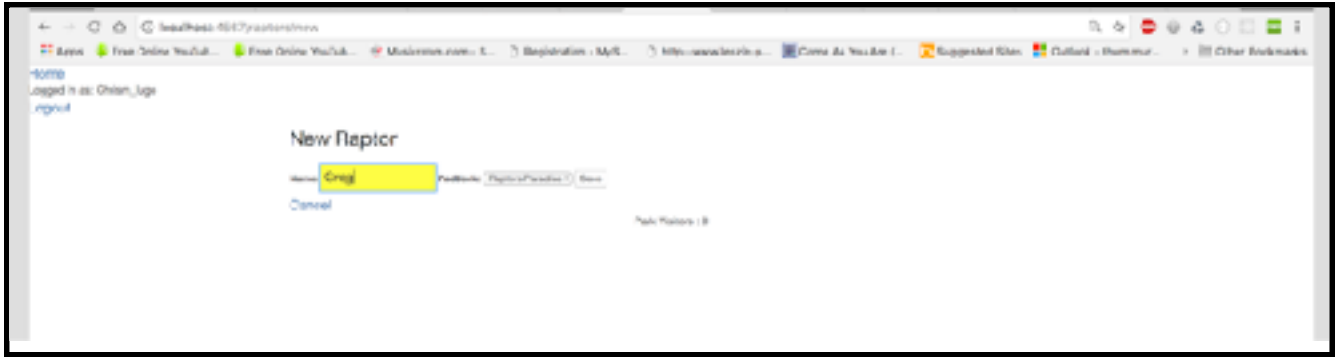
Raptors

New Raptor
Back

Name	Species	Status	Paddock			
Echo	CARNIVORE	Raging: Check Raptors Paradise food store	Raptors Paradise	View	edit	Delete
Delta	CARNIVORE	Raging: Check Raptors Paradise food store	Raptors Paradise	View	edit	Delete
Charlie	CARNIVORE	Raging: Check Raptors Paradise food store	Raptors Paradise	View	edit	Delete
Blue	CARNIVORE	Raging: Check Raptors Paradise food store	Raptors Paradise	View	edit	Delete

Park Visitors : 0

Inputs the name of the new raptor and paddock and clicks save



Home
Logged in as: [illegible]
Logout

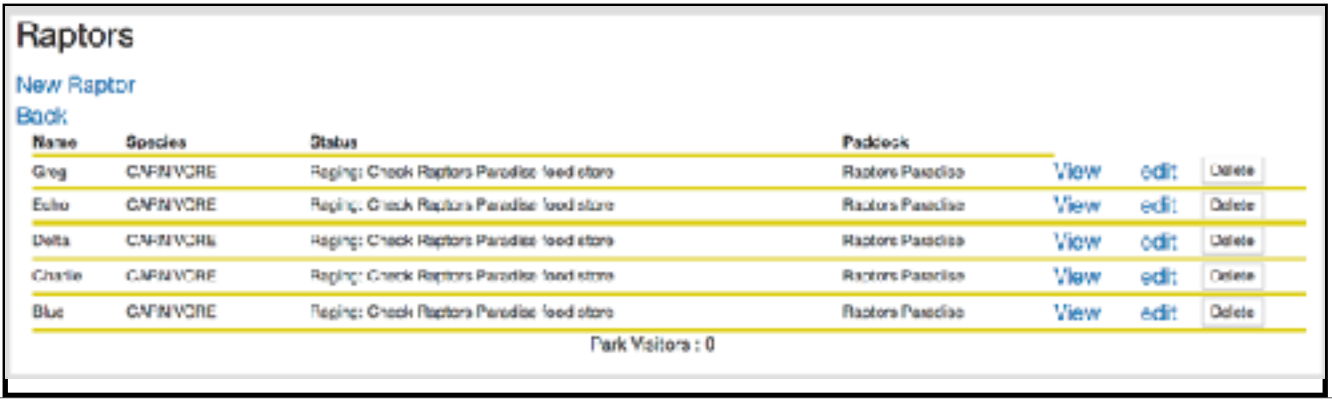
New Raptor

Name: Greg Paddock: RaptorsParadise Save

Cancel

Park Visitors : 0

The input is saved to the database which is confirmed when viewed on raptors page.



Raptors

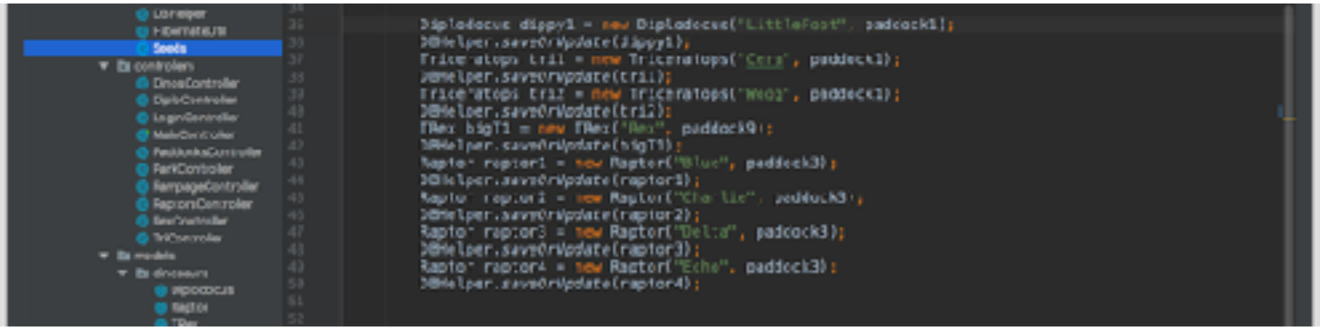
New Raptor
Back

Name	Species	Status	Paddock			
Greg	CARNIVORE	Raging: Check Raptors Paradise food store	Raptors Paradise	View	edit	Delete
Echo	CARNIVORE	Raging: Check Raptors Paradise food store	Raptors Paradise	View	edit	Delete
Delta	CARNIVORE	Raging: Check Raptors Paradise food store	Raptors Paradise	View	edit	Delete
Charlie	CARNIVORE	Raging: Check Raptors Paradise food store	Raptors Paradise	View	edit	Delete
Blue	CARNIVORE	Raging: Check Raptors Paradise food store	Raptors Paradise	View	edit	Delete

Park Visitors : 0

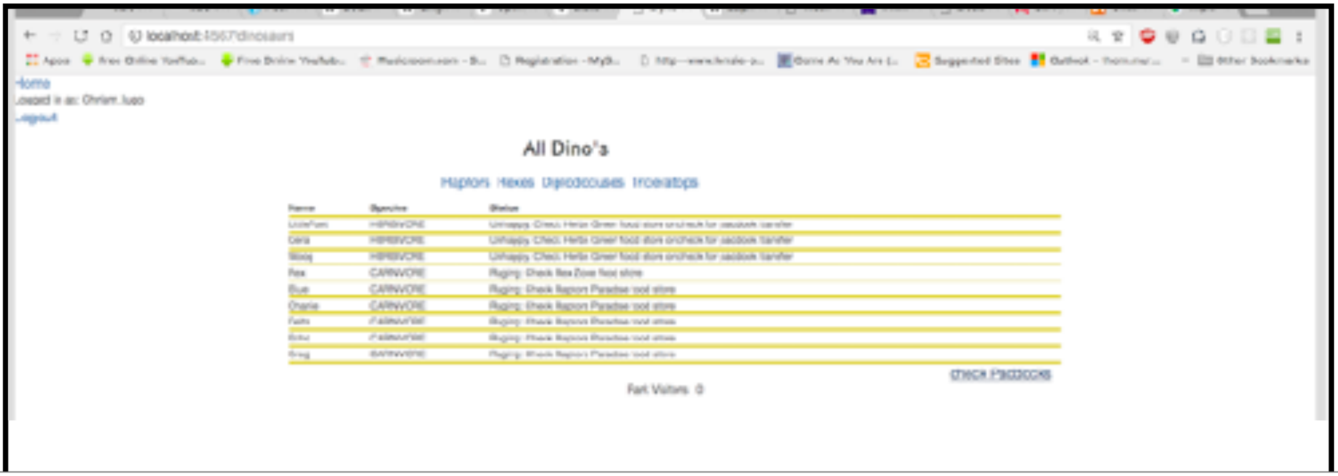
P	P 14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved	

This diagram shows a seeds file which on launch of the application stores this set of dinosaurs to the database



```
Diplodocus dippy1 = new Diplodocus("LittleFoot", paddock1);
DBHelper.saveOrUpdate(dippy1);
Triceratops tri1 = new Triceratops("Cera", paddock1);
DBHelper.saveOrUpdate(tri1);
Triceratops tri2 = new Triceratops("Woog", paddock1);
DBHelper.saveOrUpdate(tri2);
TRex bigT1 = new TRex("Rex", paddock9);
DBHelper.saveOrUpdate(bigT1);
Raptor raptor1 = new Raptor("Blue", paddock3);
DBHelper.saveOrUpdate(raptor1);
Raptor raptor2 = new Raptor("Charlie", paddock3);
DBHelper.saveOrUpdate(raptor2);
Raptor raptor3 = new Raptor("Delta", paddock3);
DBHelper.saveOrUpdate(raptor3);
Raptor raptor4 = new Raptor("Echo", paddock3);
DBHelper.saveOrUpdate(raptor4);
```

Here we can see the data is persisted.



Logged in as: Chris.luo

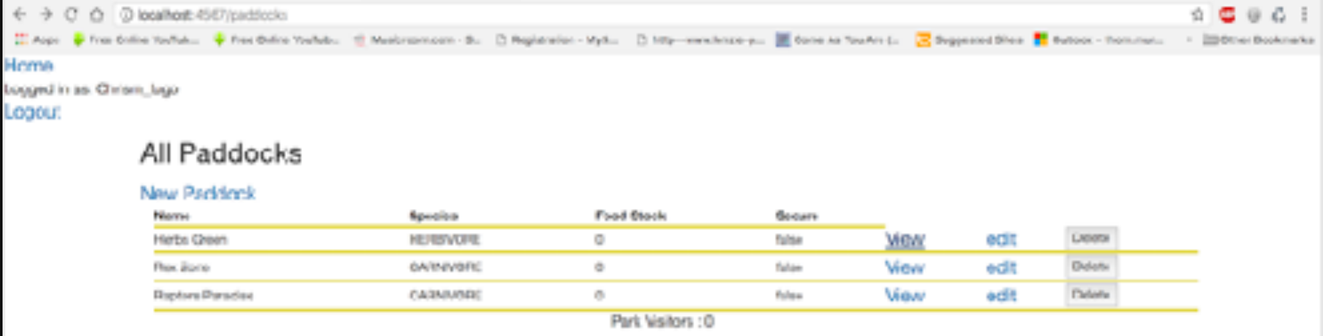
All Dino's

Name	Species	Status
LittleFoot	HERBIVORE	Unhappy. Check Herbs Green food store or check for paddock transfer
Cera	HERBIVORE	Unhappy. Check Herbs Green food store or check for paddock transfer
Woog	HERBIVORE	Unhappy. Check Herbs Green food store or check for paddock transfer
Rex	CARNIVORE	Raging. Check Rex Zone food store
Blue	CARNIVORE	Raging. Check Raptor Paradise food store
Charlie	CARNIVORE	Raging. Check Raptor Paradise food store
Delta	CARNIVORE	Raging. Check Raptor Paradise food store
Echo	CARNIVORE	Raging. Check Raptor Paradise food store
Greg	CARNIVORE	Raging. Check Raptor Paradise food store

Park Visitors: 0

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	

User clicks view to see details for Herbs Green



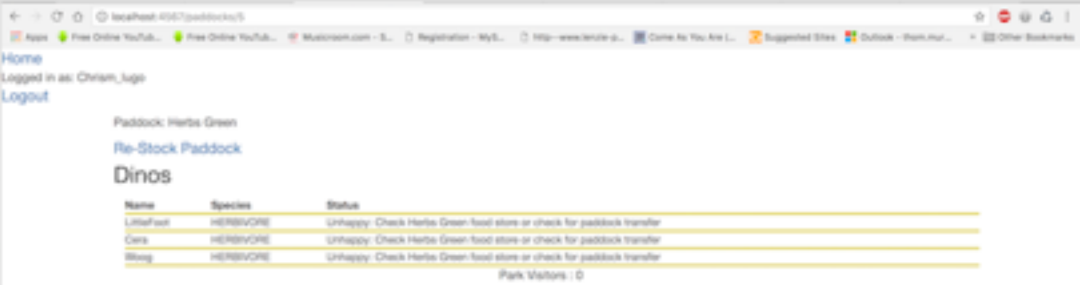
All Paddocks

New Paddock

Name	Species	Food Stock	Secure			
Herbs Green	HERBIVORE	0	false	View	edit	Delete
Rex Zone	CARNIVORE	0	false	View	edit	Delete
Raptors Paradise	CARNIVORE	0	false	View	edit	Delete

Park Visitors :0

Link Navigates to paddock page for Herb’s Green



Paddock: Herbs Green

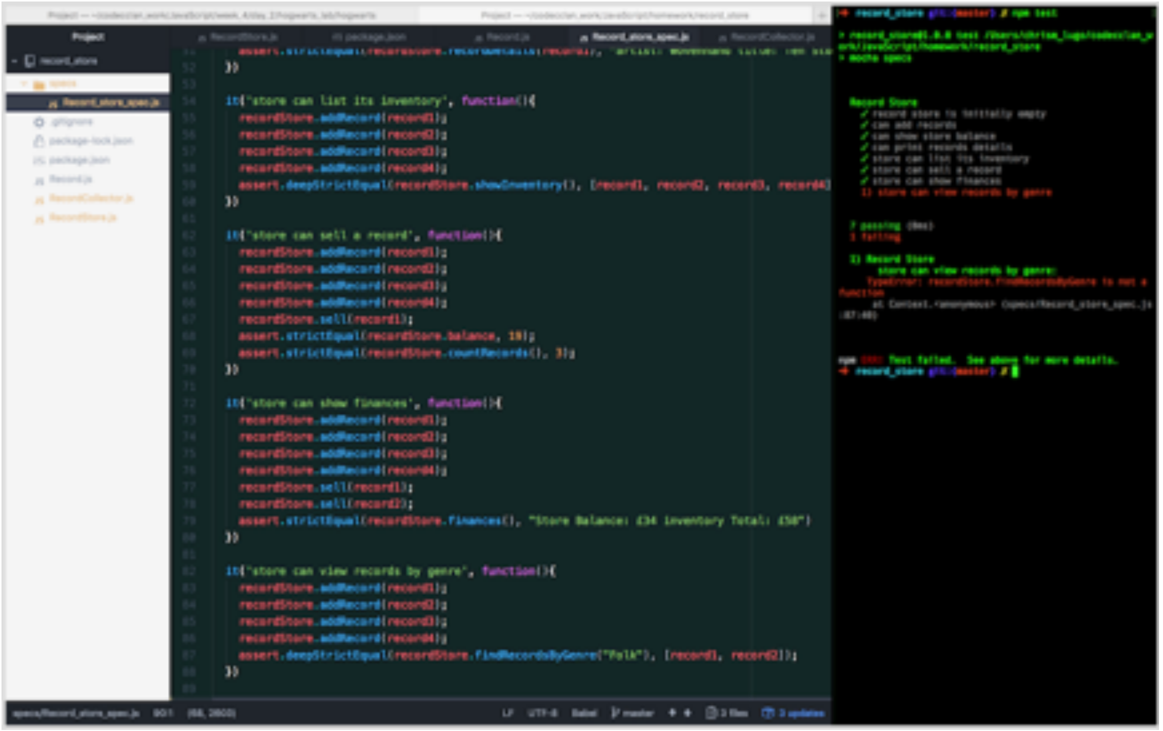
Re-Stock Paddock

Dinos

Name	Species	Status
LittleFoot	HERBIVORE	Unhappy: Check Herbs Green food store or check for paddock transfer
Cera	HERBIVORE	Unhappy: Check Herbs Green food store or check for paddock transfer
Woog	HERBIVORE	Unhappy: Check Herbs Green food store or check for paddock transfer

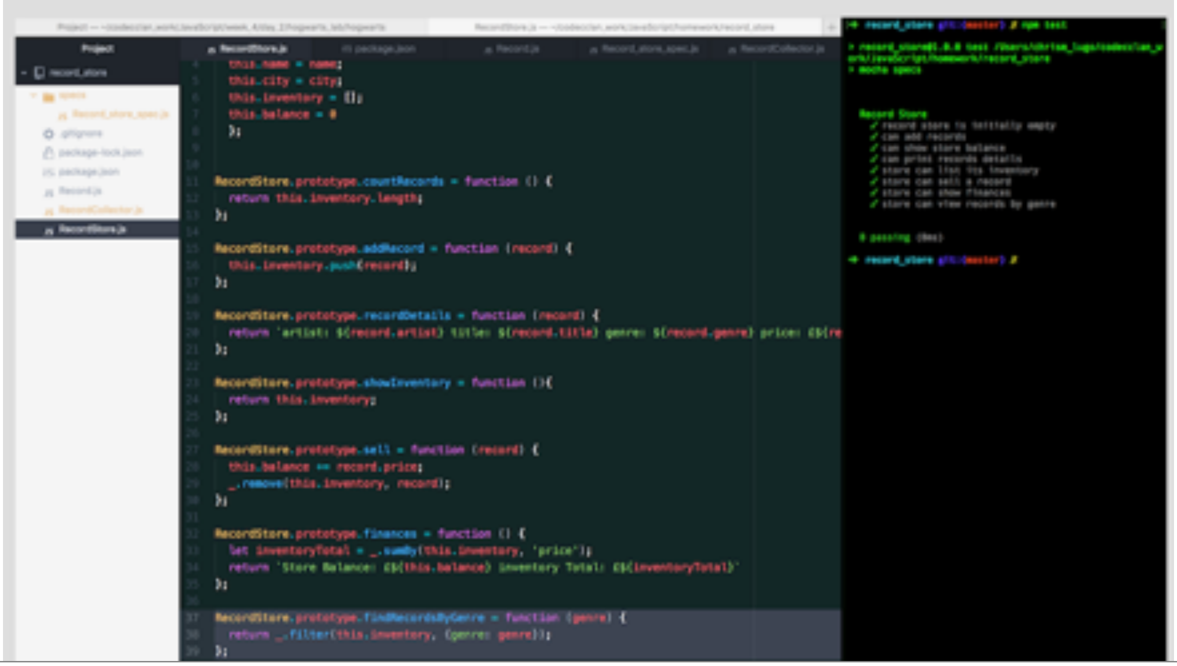
Park Visitors : 0

	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	



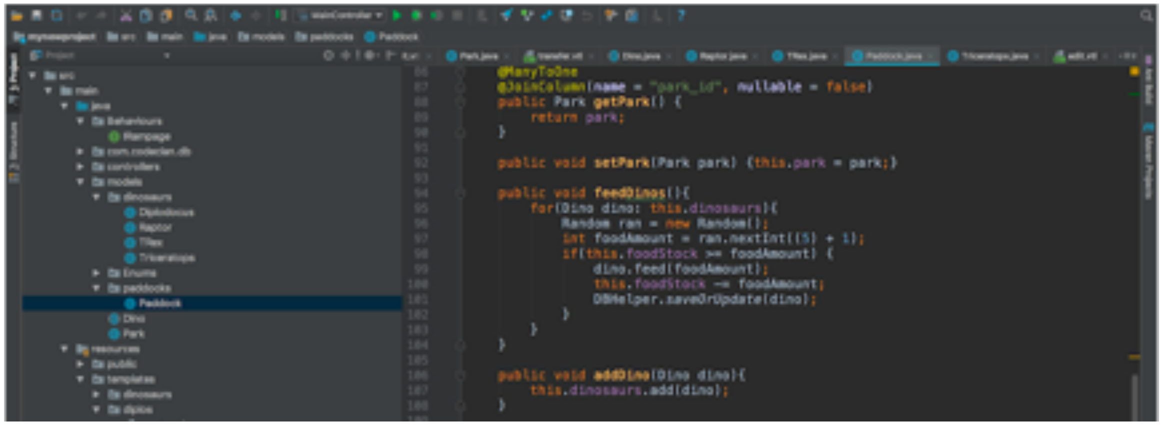
```
  })

  it('store can list its inventory', function(){
    recordStore.addRecord(record1);
    recordStore.addRecord(record2);
    recordStore.addRecord(record3);
    recordStore.addRecord(record4);
    assert.deepStrictEqual(recordStore.showInventory(), [record1, record2, record3, record4]
  })

  it('store can sell a record', function(){
    recordStore.addRecord(record1);
    recordStore.addRecord(record2);
    recordStore.addRecord(record3);
    recordStore.addRecord(record4);
    recordStore.sell(record1);
    assert.strictEqual(recordStore.balance, 10);
    assert.strictEqual(recordStore.countRecords(), 3);
  })

  it('store can show finances', function(){
    recordStore.addRecord(record1);
    recordStore.addRecord(record2);
    recordStore.addRecord(record3);
    recordStore.addRecord(record4);
    recordStore.sell(record1);
    recordStore.sell(record2);
    assert.strictEqual(recordStore.finances(), "Store Balance: £34 inventory Total: £50")
  })

  it('store can view records by genre', function(){
    recordStore.addRecord(record1);
    recordStore.addRecord(record2);
    recordStore.addRecord(record3);
    recordStore.addRecord(record4);
    assert.deepStrictEqual(recordStore.findRecordsByGenre("Folk"), [record1, record2]);
  })
```

```
→ record_store git:(master) ✗ npm test

> record_store@1.0.0 test /Users/chrisw_lugo/codeclan_w
ork/javaScript/homework/record_store
> mocha specs



  Record Store
    ✓ record store is initially empty
    ✓ can add records
    ✓ can show store balance
    ✓ can print records details
    ✓ store can list its inventory
    ✓ store can sell a record
    ✓ store can show finances
    1) store can view records by genre


  7 passing (8ms)
  1 failing

  1) Record Store
       store can view records by genre:
     TypeError: recordStore.findRecordsByGenre is not a
function
      at Context.<anonymous> (specs/Record_store_spec.js
:87:40)



npm ERR! Test failed.  See above for more details.
→ record_store git:(master) ✗ 
```

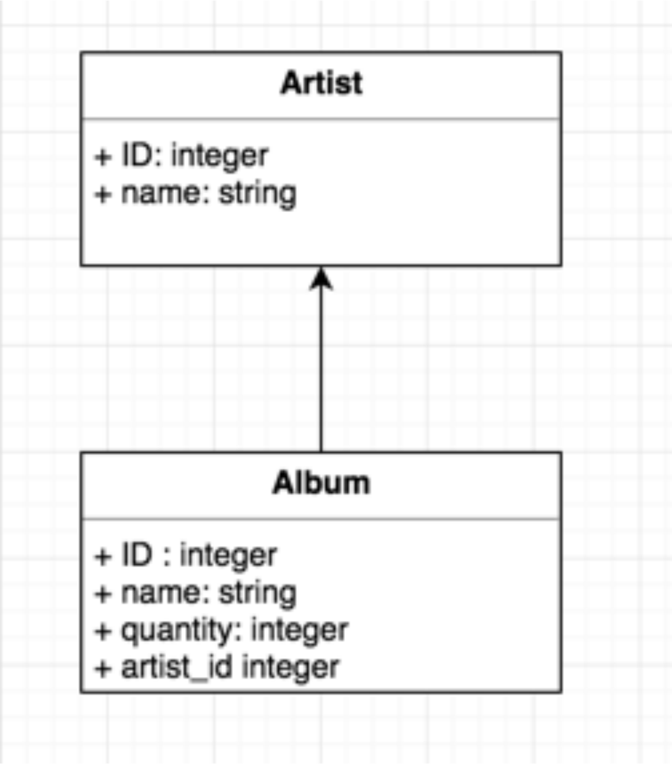
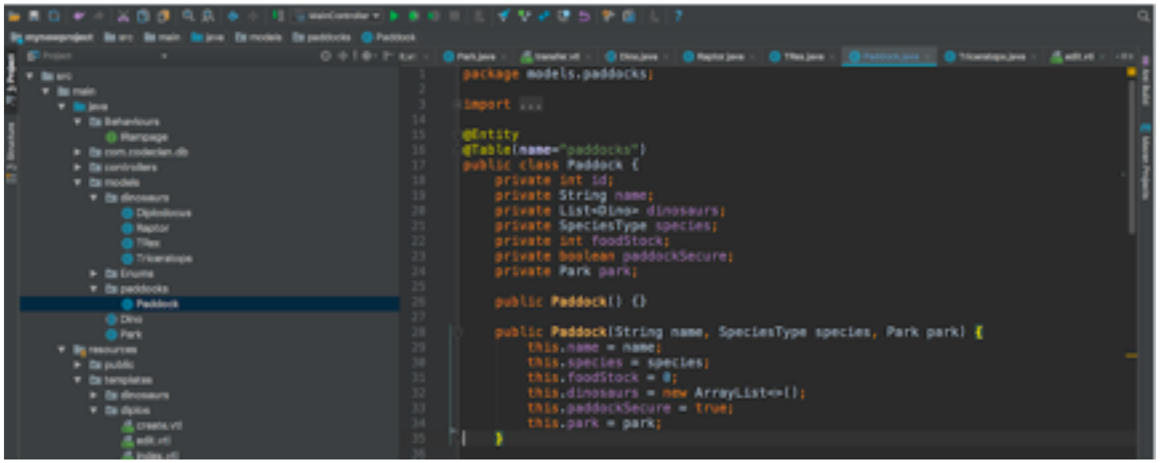


```
  this.name = name;
  this.city = city;
  this.inventory = [];
  this.balance = 0
  };


RecordStore.prototype.countRecords = function () {
  return this.inventory.length;
};

RecordStore.prototype.addRecord = function (record) {
  this.inventory.push(record);
};

RecordStore.prototype.recordDetails = function (record) {
  return `artist: ${record.artist} title: ${record.title} genre: ${record.genre} price: £${re
};

RecordStore.prototype.showInventory = function (){
  return this.inventory;
};

RecordStore.prototype.sell = function (record) {
  this.balance += record.price;
  _.remove(this.inventory, record);
};

RecordStore.prototype.finances = function () {
  let inventoryTotal = _.sumBy(this.inventory, 'price');
  return `Store Balance: £${this.balance} inventory Total: £${inventoryTotal}`
};

RecordStore.prototype.findRecordsByGenre = function (genre) {
  return _.filter(this.inventory, {genre: genre});
};
```

```
→ record_store git:(master) ✗ npm test

> record_store@1.0.0 test /Users/chrisw_lugo/codeclan_w
ork/javaScript/homework/record_store
> mocha specs



  Record Store
    ✓ record store is initially empty
    ✓ can add records
    ✓ can show store balance
    ✓ can print records details
    ✓ store can list its inventory
    ✓ store can sell a record
    ✓ store can show finances
    ✓ store can view records by genre


  8 passing (8ms)

→ record_store git:(master) ✗
```

Unit	Ref.	Evidence	Done
I & T	I.T 7	Demonstrate the use of Polymorphism in a program.	
			
A & D	A.D 5	An Inheritance Diagram	

```
    @ManyToOne
    @JoinColumn(name = "park_id", nullable = false)
    public Park getPark() {
        return park;
    }

    public void setPark(Park park) {this.park = park;}

    public void feedDinos(){
        for(Dino dino: this.dinosaurs){
            Random ran = new Random();
            int foodAmount = ran.nextInt((5) + 1);
            if(this.foodStock >= foodAmount) {
                dino.feed(foodAmount);
                this.foodStock -= foodAmount;
                DBHelper.saveOrUpdate(dino);
            }
        }
    }

    public void addDino(Dino dino){
        this.dinosaurs.add(dino);
    }
```

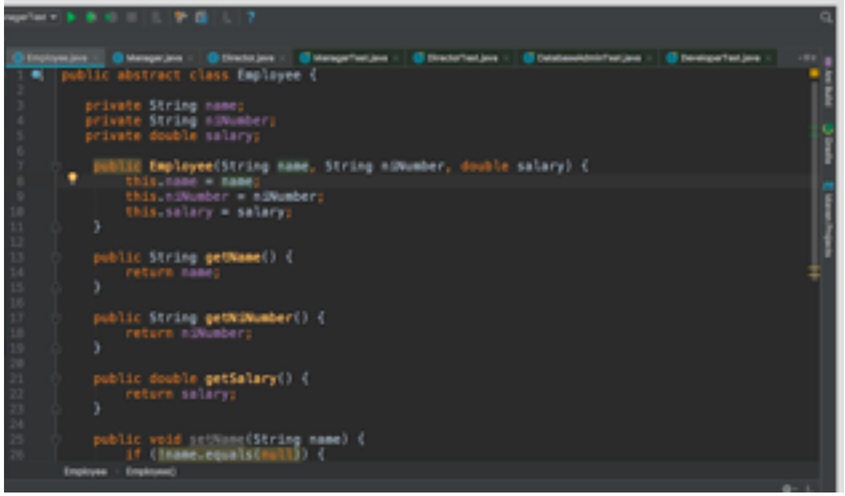
			
I & T	I.T 1	Take a screenshot of an example of encapsulation in a program.	
			

```
package models.paddocks;

import ...

@Entity
@Table(name="paddocks")
public class Paddock {
    private int id;
    private String name;
    private List<Dino> dinosaurs;
    private SpeciesType species;
    private int foodStock;
    private boolean paddockSecure;
    private Park park;

    public Paddock() {}

    public Paddock(String name, SpeciesType species, Park park) {
        this.name = name;
        this.species = species;
        this.foodStock = 0;
        this.dinosaurs = new ArrayList<>();
        this.paddockSecure = true;
        this.park = park;
    }
```

	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.	

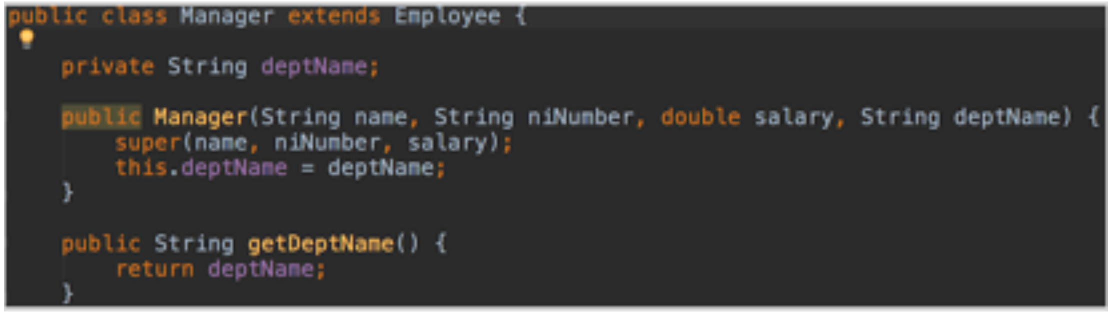
Week 7

This Employee Class contains all the functionality for all types of employee



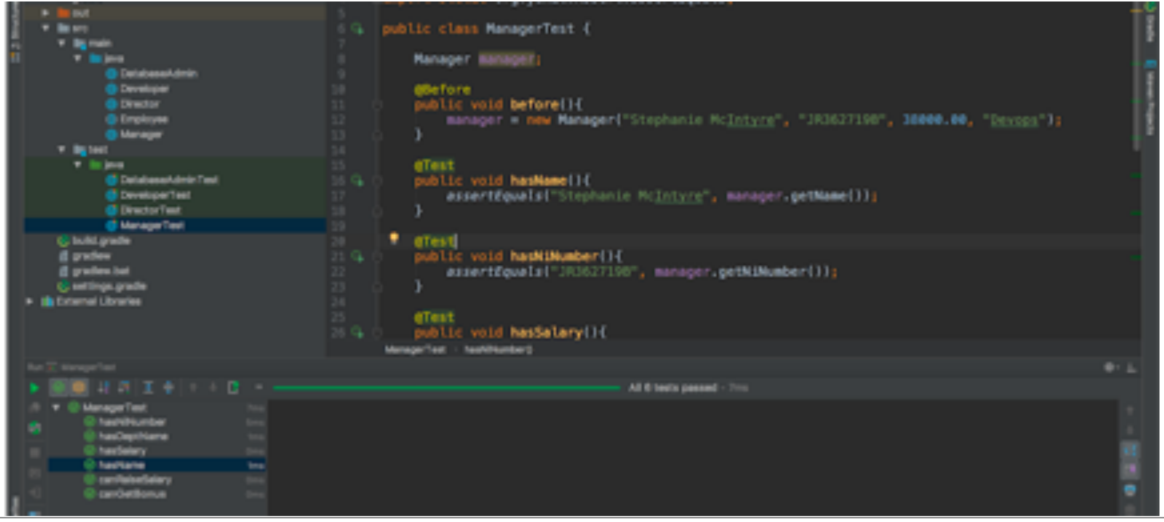
```
public abstract class Employee {

    private String name;
    private String niNumber;
    private double salary;

    public Employee(String name, String niNumber, double salary) {
        this.name = name;
        this.niNumber = niNumber;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public String getNiNumber() {
        return niNumber;
    }

    public double getSalary() {
        return salary;
    }

    public void setName(String name) {
        if (!name.equals(null)) {
```

The Manager class shown here inherits from the Employee class.



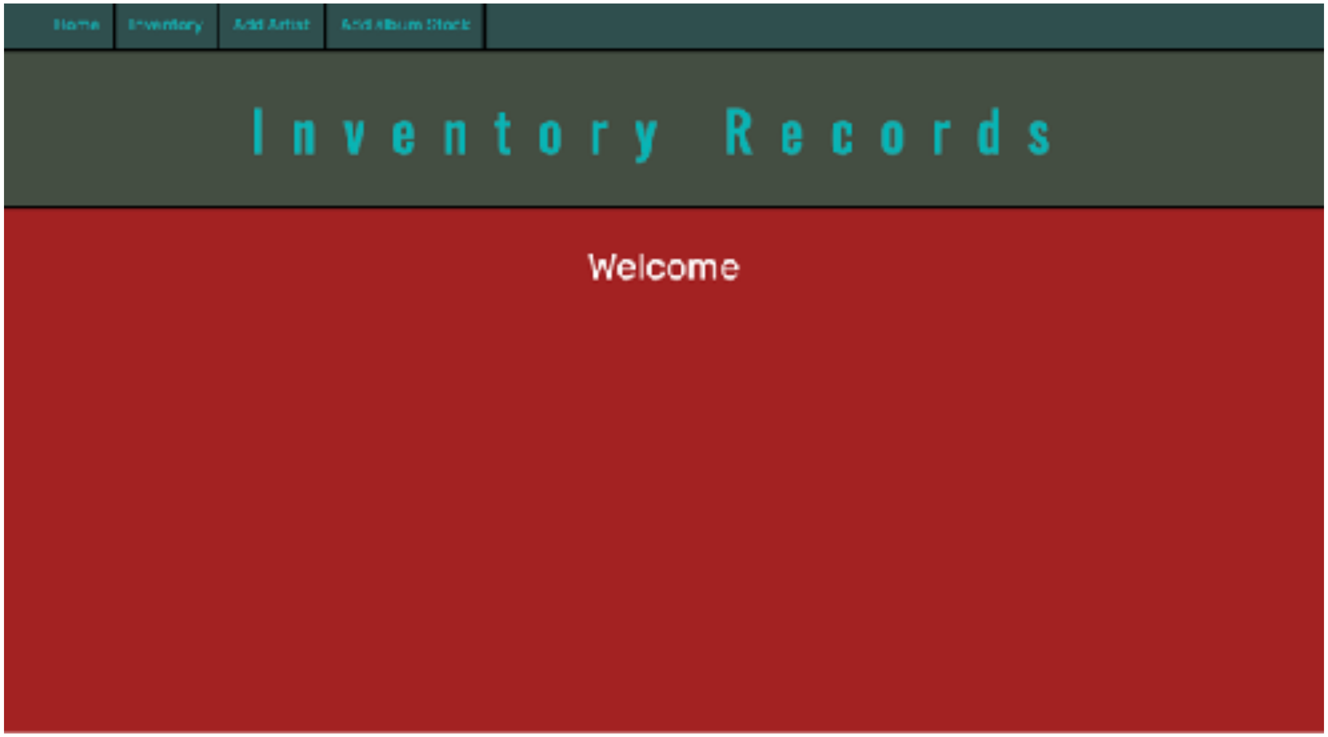
```
public class Manager extends Employee {

    private String deptName;

    public Manager(String name, String niNumber, double salary, String deptName) {
        super(name, niNumber, salary);
        this.deptName = deptName;
    }

    public String getDeptName() {
        return deptName;
    }
```

The test file here demonstrates the manager Class has inherited the methods of its parent class.

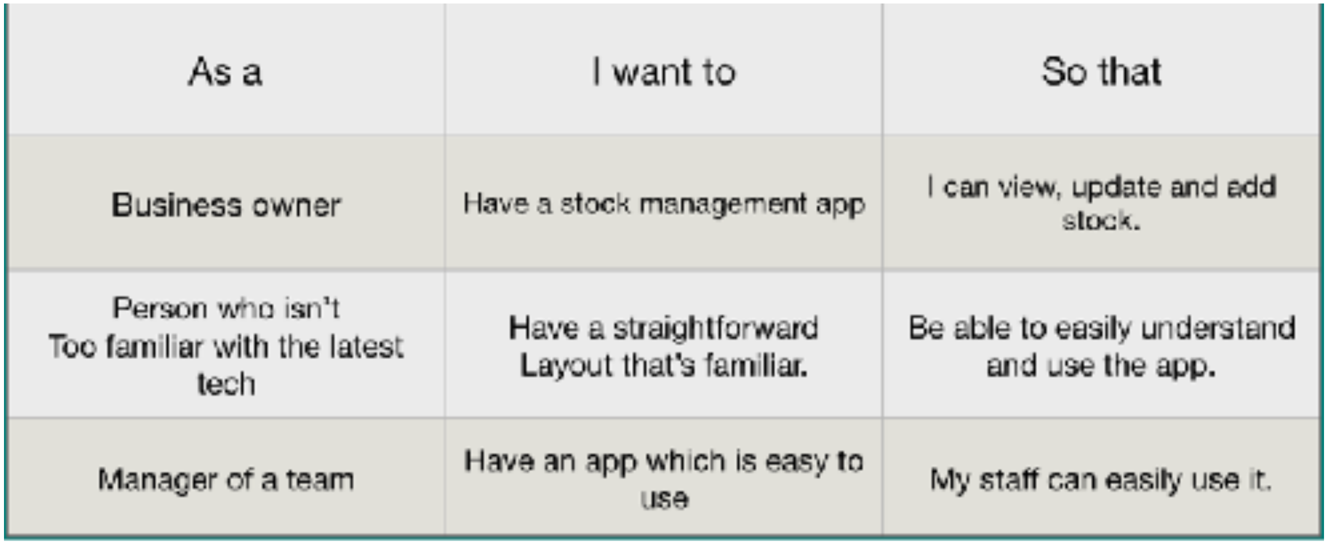


```
public class ManagerTest {

    Manager manager;

    @Before
    public void before(){
        manager = new Manager("Stephanie McIntyre", "JR3627198", 38000.00, "Devops");
    }

    @Test
    public void hasName(){
        assertEquals("Stephanie McIntyre", manager.getName());
    }

    @Test
    public void hasNiNumber(){
        assertEquals("JR3627198", manager.getNiNumber());
    }

    @Test
    public void hasSalary(){
```

```
All 6 tests passed - 7ms

ManagerTest
    hasNiNumber
    hasDeptName
    hasSalary
    hasName
    canRaiseSalary
    canGetBonus
```

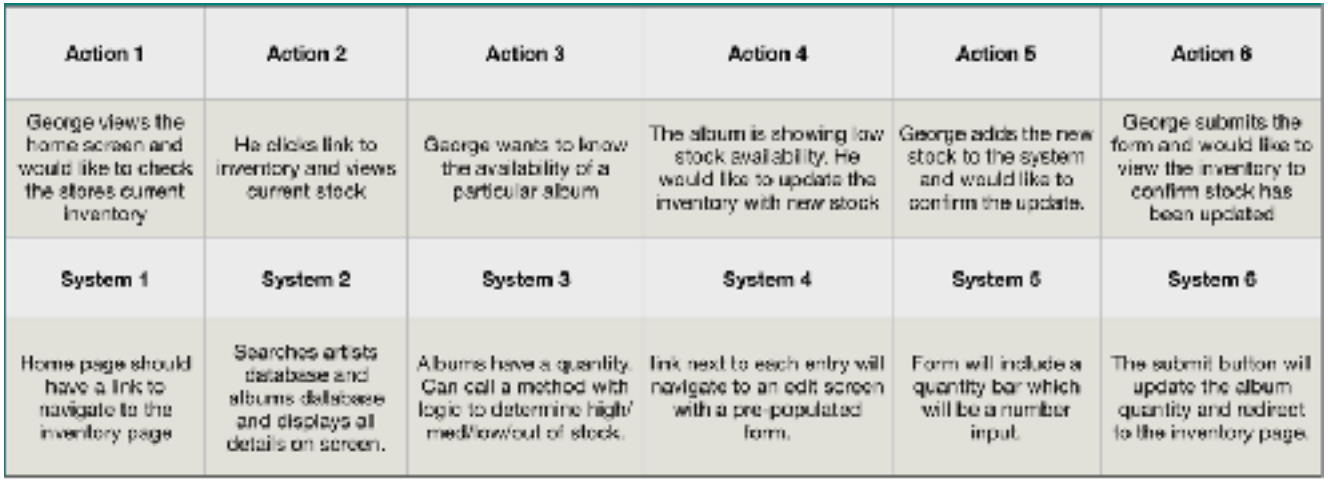
	P	P 11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.	
			 Home Inventory Add Artist Add Album Stock Inventory Records Welcome https://github.com/Chrismlugo/Project-1	
	P	P 12	Take screenshots or photos of your planning and the different stages of development to show changes.	

User Needs

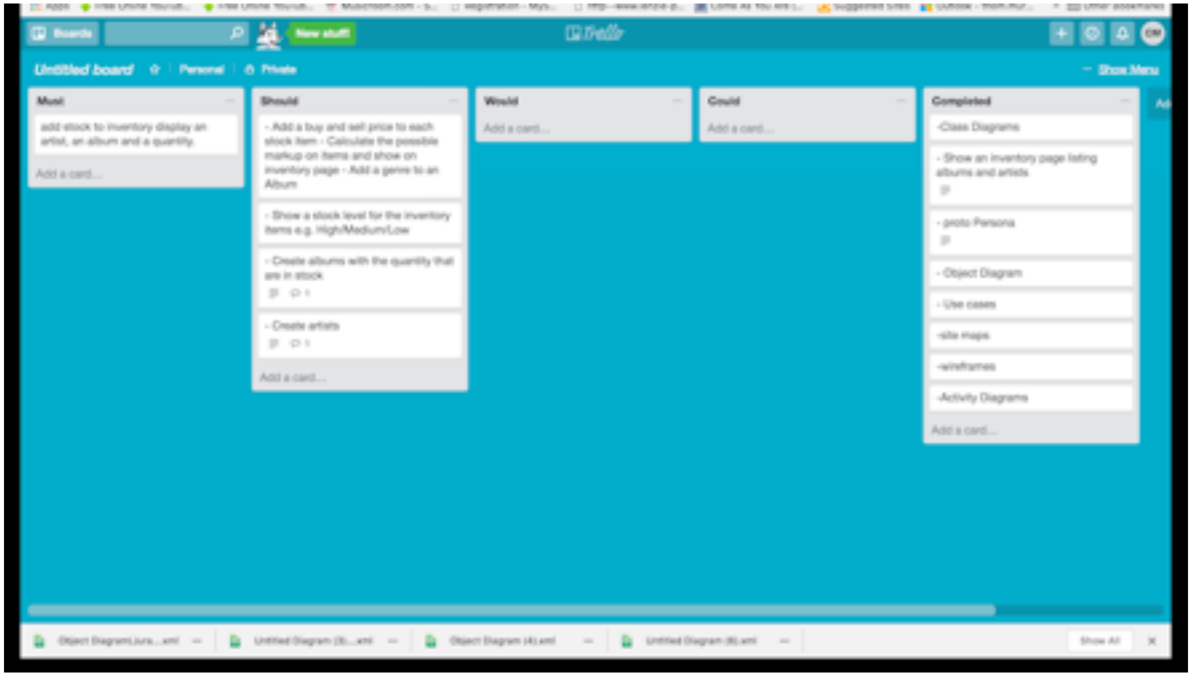
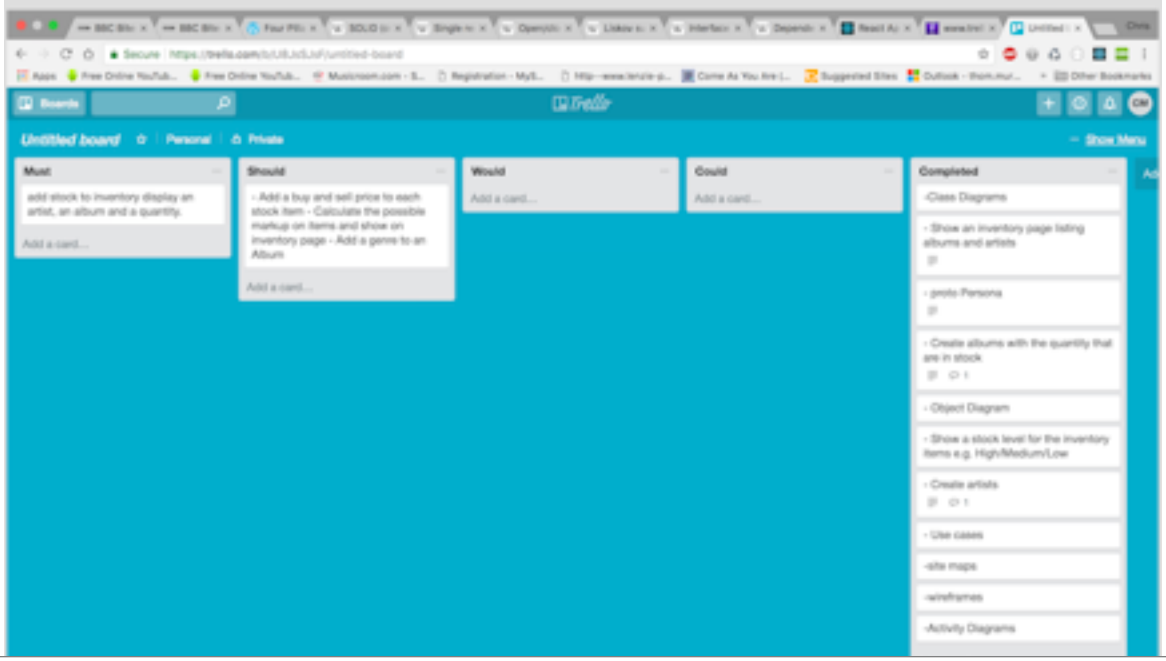


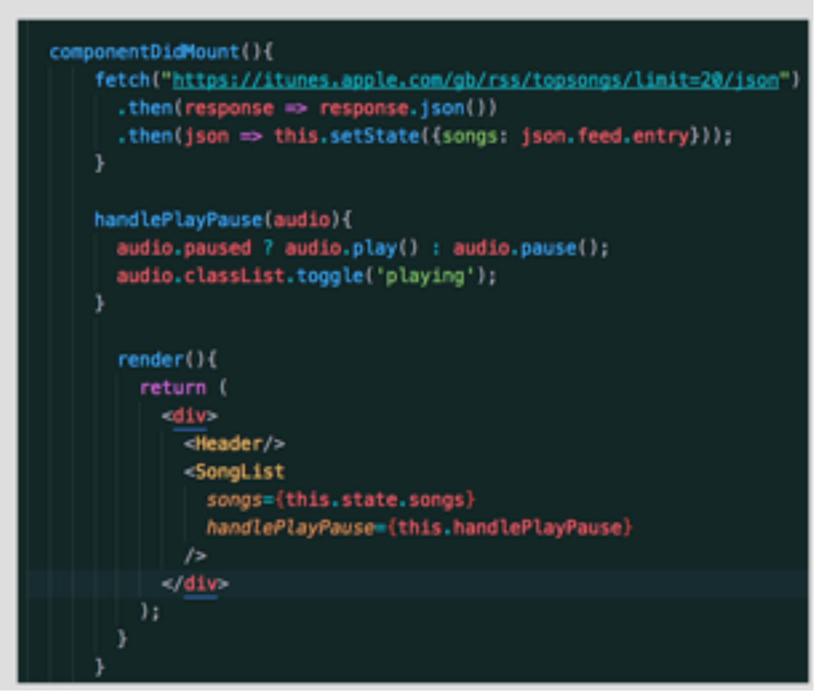
As a	I want to	So that
Business owner	Have a stock management app	I can view, update and add stock.
Person who isn't Too familiar with the latest tech	Have a straightforward Layout that's familiar.	Be able to easily understand and use the app.
Manager of a team	Have an app which is easy to use	My staff can easily use it.

User Journey

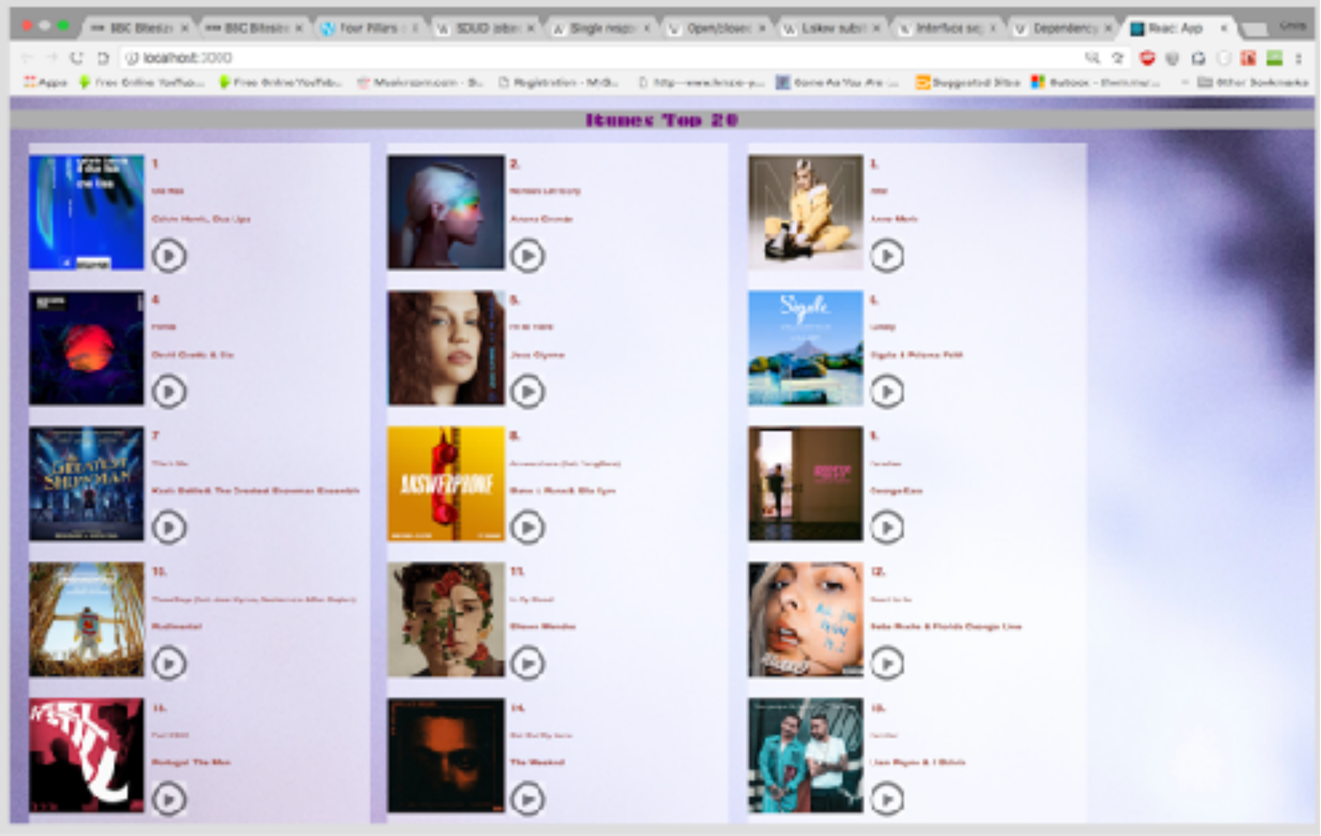


Action 1	Action 2	Action 3	Action 4	Action 5	Action 6
George views the home screen and would like to check the stores current inventory	He clicks link to inventory and views current stock	George wants to know the availability of a particular album	The album is showing low stock availability. He would like to update the inventory with new stock	George adds the new stock to the system and would like to confirm the update.	George submits the form and would like to view the inventory to confirm stock has been updated
System 1	**System 2**	**System 3**	**System 4**	**System 5**	**System 6**
Home page should have a link to navigate to the inventory page	Searches artists database and albums database and displays all details on screen.	Albums have a quantity. Can call a method with logic to determine high/med/low/out of stock.	link next to each entry will navigate to an edit screen with a pre-populated form.	Form will include a quantity bar which will be a number input.	The submit button will update the album quantity and redirect to the inventory page.

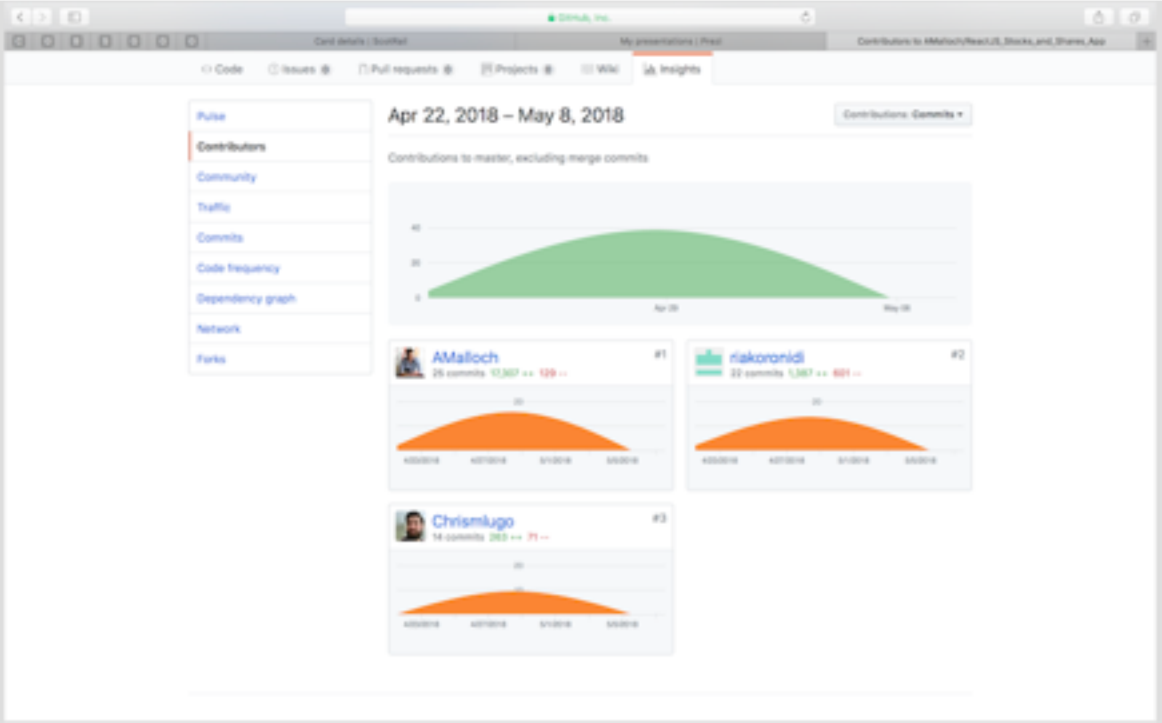
			 Untitled board Must add stock to inventory display an artist, an album and a quantity. Should - Add a buy and sell price to each stock item - Calculate the possible markup on items and show on inventory page - Add a genre to an Album - Show a stock level for the inventory items e.g. High/Medium/Low - Create albums with the quantity that are in stock - Create artists Would Could Completed -Class Diagrams - Show an inventory page listing albums and artists - proto Persona - Object Diagram - Use cases -site maps -wireframes -Activity Diagrams  Untitled board Must add stock to inventory display an artist, an album and a quantity. Should - Add a buy and sell price to each stock item - Calculate the possible markup on items and show on inventory page - Add a genre to an Album Would Could Completed -Class Diagrams - Show an inventory page listing albums and artists - proto Persona - Create albums with the quantity that are in stock - Object Diagram - Show a stock level for the inventory items e.g. High/Medium/Low - Create artists - Use cases -site maps -wireframes -Activity Diagrams	

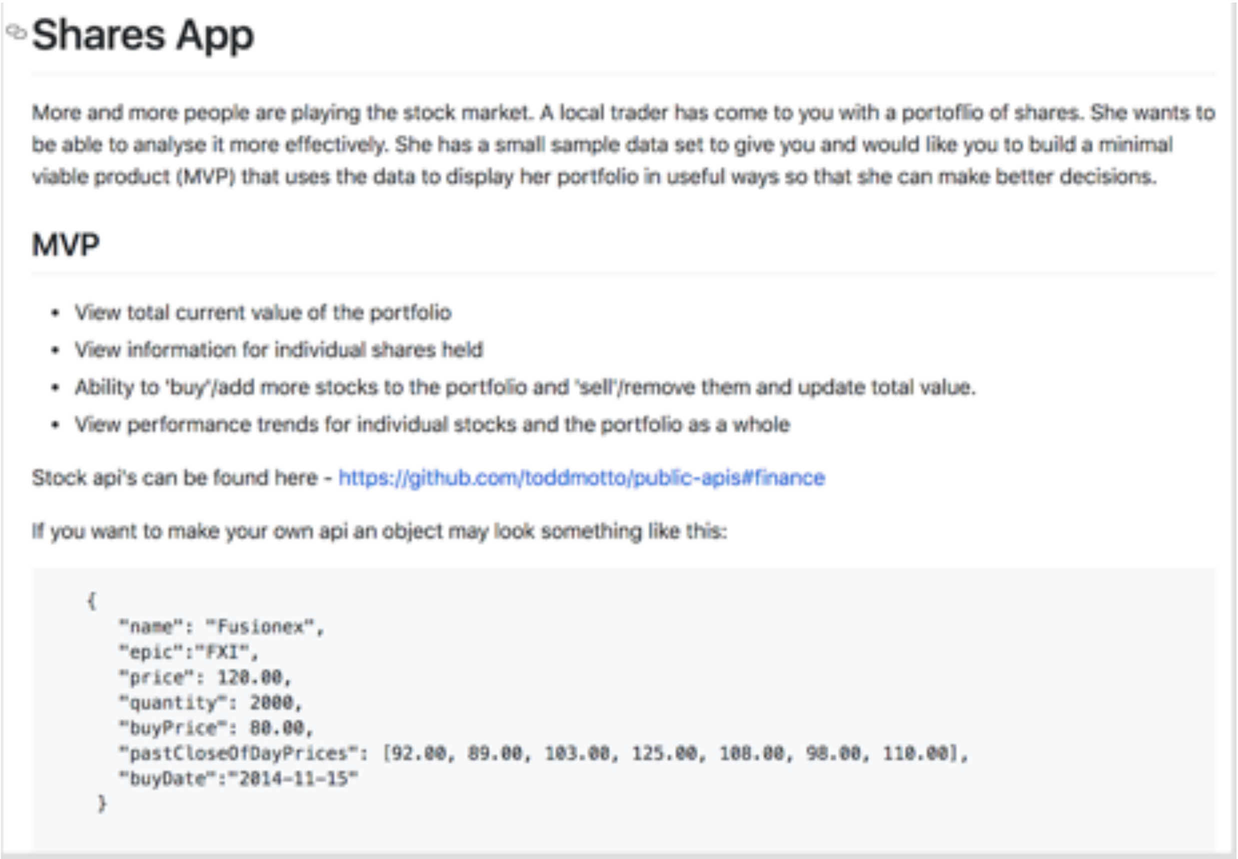
Week 11	Unit	Ref.	Evidence	Done
	I & T		Unit, integration and acceptance testing task B	
			https://github.com/Chrismlugo/calculator_unit_and_integration_testing_mocha_selenium	
	P	P 16	Show an API being used within your program. Take a screenshot of: * The code that uses or implements the API * The API being used by the program whilst running	
				

```
componentDidMount(){
    fetch("https://itunes.apple.com/gb/rss/topsongs/limit=20/json")
      .then(response => response.json())
      .then(json => this.setState({songs: json.feed.entry}));
    }

    handlePlayPause(audio){
      audio.paused ? audio.play() : audio.pause();
      audio.classList.toggle('playing');
    }

      render(){
        return (
          <div>
            <Header/>
            <SongList
              songs={this.state.songs}
              handlePlayPause={this.handlePlayPause}
            />
          </div>
        );
      }
    }
```

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

		 Apr 22, 2018 – May 8, 2018 Contributions to master, excluding merge commits Code Issues Pull requests Projects Wiki Insights Pulse Contributors Community Traffic Commits Code frequency Dependency graph Network Forks Contributions: Commits AMalloch #1 25 commits riakoronidi #2 22 commits Chrismlugo #3 14 commits	
P	P 2	Take a screenshot of the project brief from your group project.	

<table>
<tr>
<td></td>
<td></td>
<td>


Shares App

More and more people are playing the stock market. A local trader has come to you with a portoflio of shares. She wants to be able to analyse it more effectively. She has a small sample data set to give you and would like you to build a minimal viable product (MVP) that uses the data to display her portfolio in useful ways so that she can make better decisions.

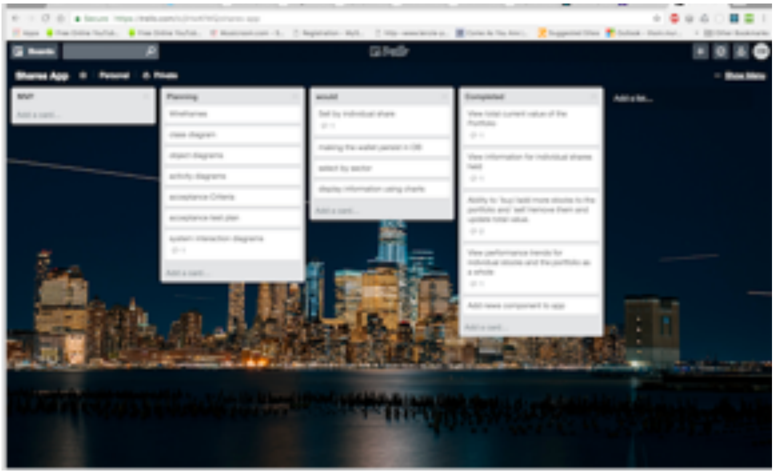
MVP

- View total current value of the portfolio
- View information for individual shares held
- Ability to 'buy'/add more stocks to the portfolio and 'sell'/remove them and update total value.
- View performance trends for individual stocks and the portfolio as a whole

Stock api's can be found here - https://github.com/toddmotto/public-apis#finance

If you want to make your own api an object may look something like this:

```
 {
   "name": "Fusionex",
   "epic":"FXI",
   "price": 120.00,
   "quantity": 2000,
   "buyPrice": 80.00,
   "pastCloseOfDayPrices": [92.00, 89.00, 103.00, 125.00, 108.00, 98.00, 110.00],
   "buyDate":"2014-11-15"
  }
```
</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>P 3</td>
<td>Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.</td>
<td></td>
</tr>
</table>



Shares App · Personal · Private

WIP

Add a card...

Planning

- Wireframes
- class diagram
- object diagrams
- activity diagrams
- acceptance Criteria
- acceptance test plan
- system interaction diagrams

Add a card...

would

- Sell by individual share
- making the wallet persist in DB
- select by sector
- display information using charts

Add a card...

Completed

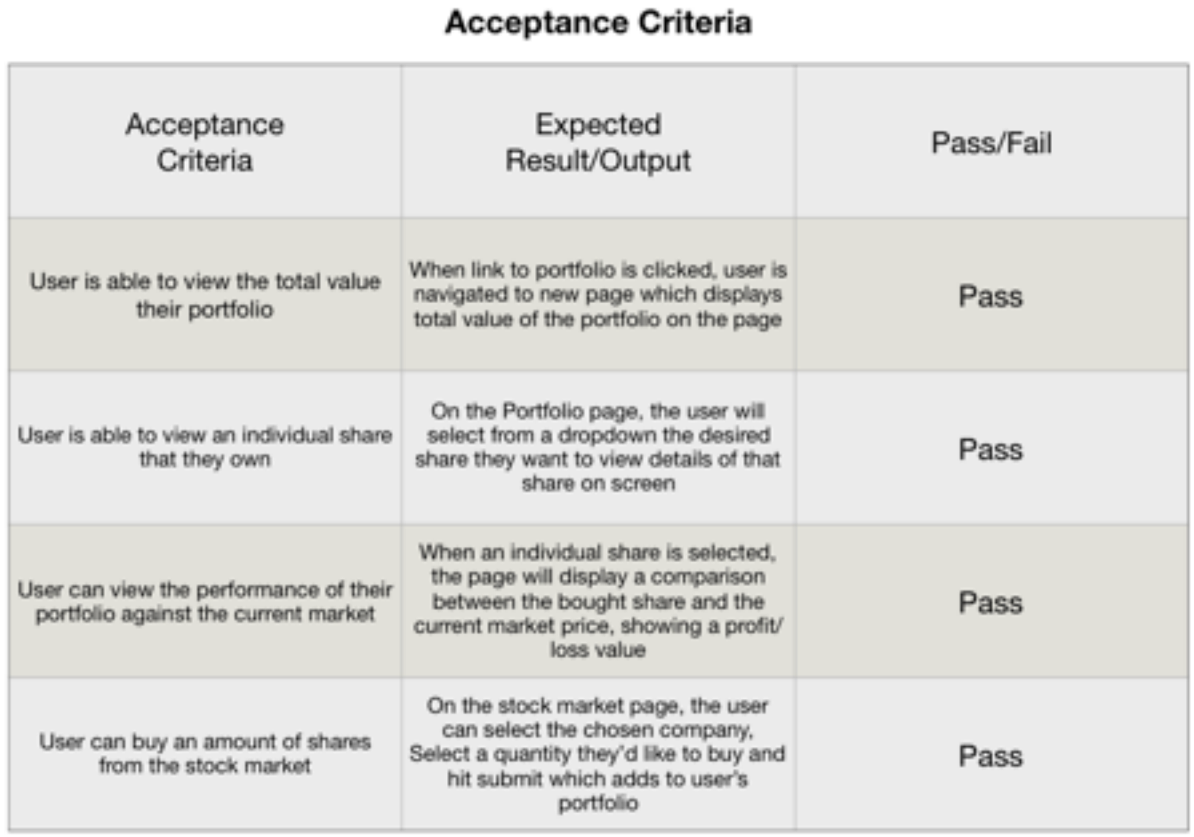
- View total current value of the Portfolio
- View information for individual shares held
- Ability to 'buy'/add more stocks to the portfolio and 'sell'/remove them and update total value.
- View performance trends for individual stocks and the portfolio as a whole
- Add news component to app

Add a card...

Add a list...

APP
MAIN
NEWS
PORTFOLIO
STOCK INFO
DB

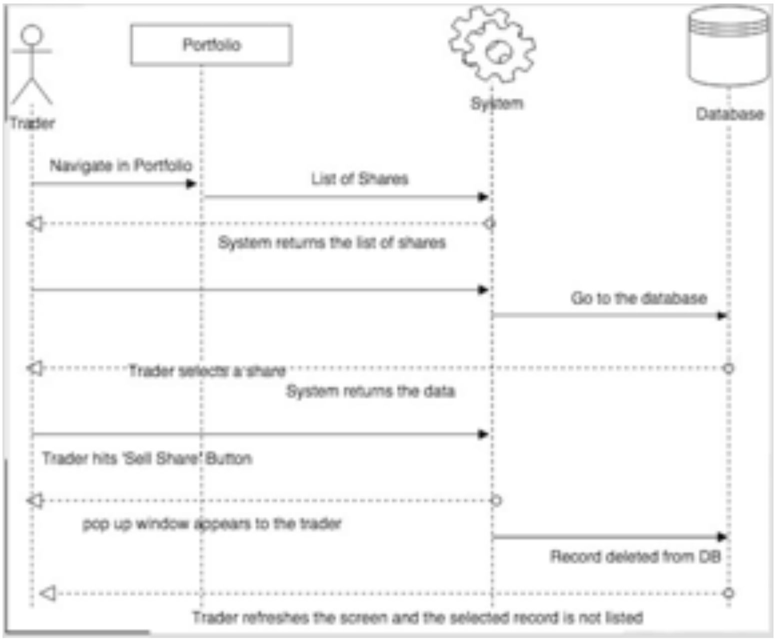
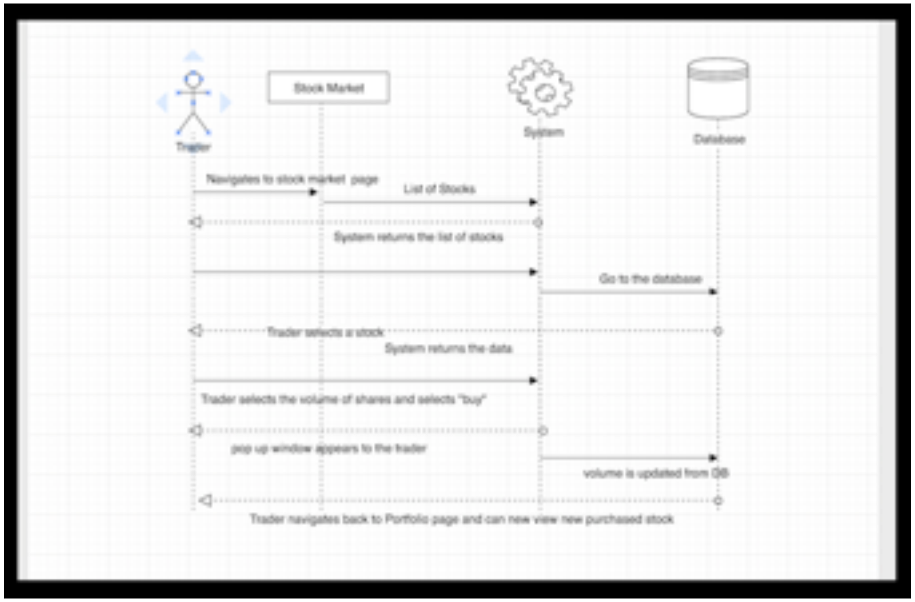
	P	P 4	Write an acceptance criteria and test plan.	

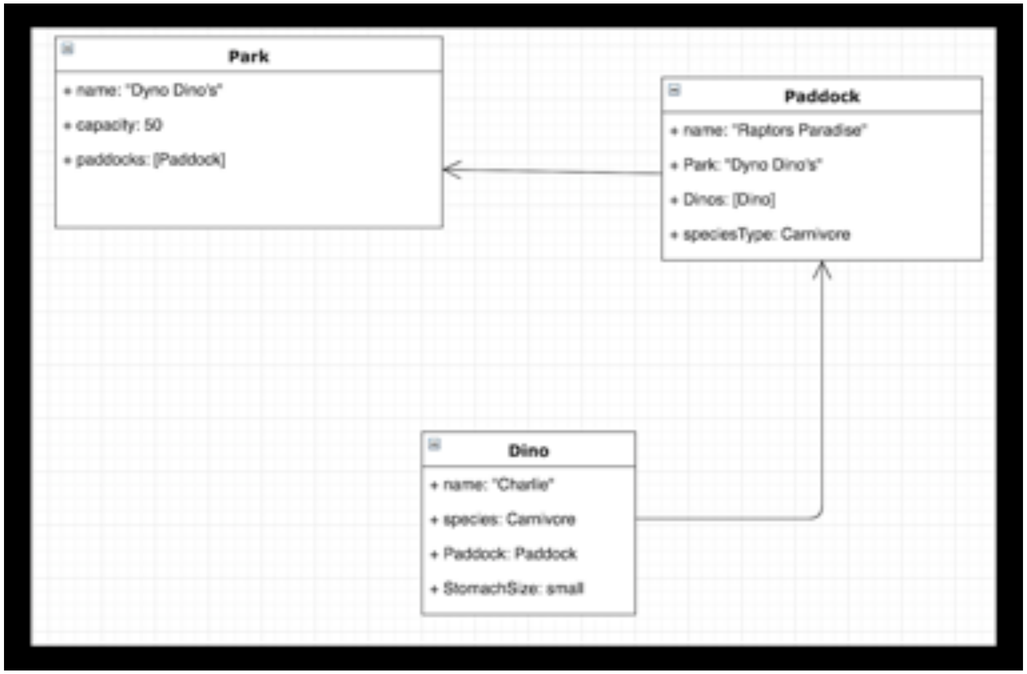


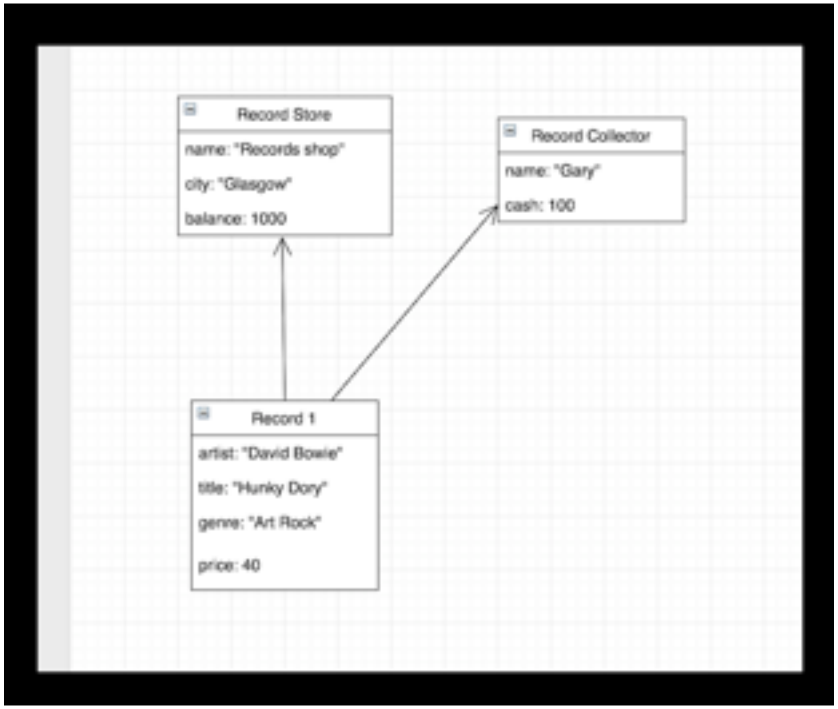
Acceptance Criteria

Acceptance Criteria	Expected Result/Output	Pass/Fail
User is able to view the total value their portfolio	When link to portfolio is clicked, user is navigated to new page which displays total value of the portfolio on the page	Pass
User is able to view an individual share that they own	On the Portfolio page, the user will select from a dropdown the desired share they want to view details of that share on screen	Pass
User can view the performance of their portfolio against the current market	When an individual share is selected, the page will display a comparison between the bought share and the current market price, showing a profit/ loss value	Pass
User can buy an amount of shares from the stock market	On the stock market page, the user can select the chosen company, Select a quantity they'd like to buy and hit submit which adds to user's portfolio	Pass

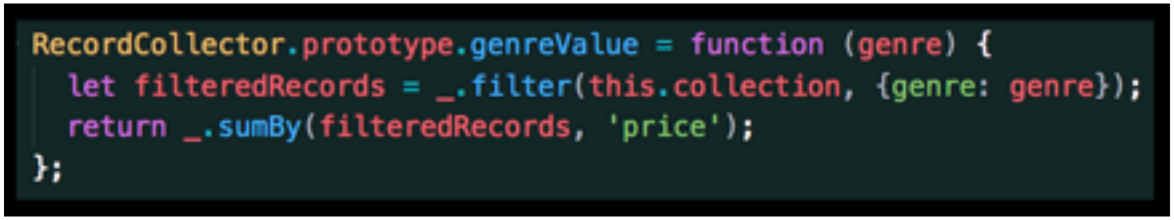
Week 13	P	P 7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).	

		 Portfolio Trader System Database Navigate in Portfolio List of Shares System returns the list of shares Go to the database Trader selects a share System returns the data Trader hits 'Sell Share' Button pop up window appears to the trader Record deleted from DB Trader refreshes the screen and the selected record is not listed  Stock Market Trader System Database Navigates to stock market page List of Stocks System returns the list of stocks Go to the database Trader selects a stock System returns the data Trader selects the volume of shares and selects "buy" pop up window appears to the trader volume is updated from DB Trader navigates back to Portfolio page and can new view new purchased stock	
P	P 8	Produce two object diagrams.	


Park
+ name: "Dyno Dino's"
+ capacity: 50
+ paddocks: [Paddock]
Paddock
+ name: "Raptors Paradise"
+ Park: "Dyno Dino's"
+ Dinos: [Dino]
+ speciesType: Carnivore
Dino
+ name: "Charlie"
+ species: Carnivore
+ Paddock: Paddock
+ StomachSize: small


Record Store
name: "Records shop"
city: "Glasgow"
balance: 1000
Record Collector
name: "Gary"
cash: 100
Record 1
artist: "David Bowie"
title: "Hunky Dory"
genre: "Art Rock"
price: 40

	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.	

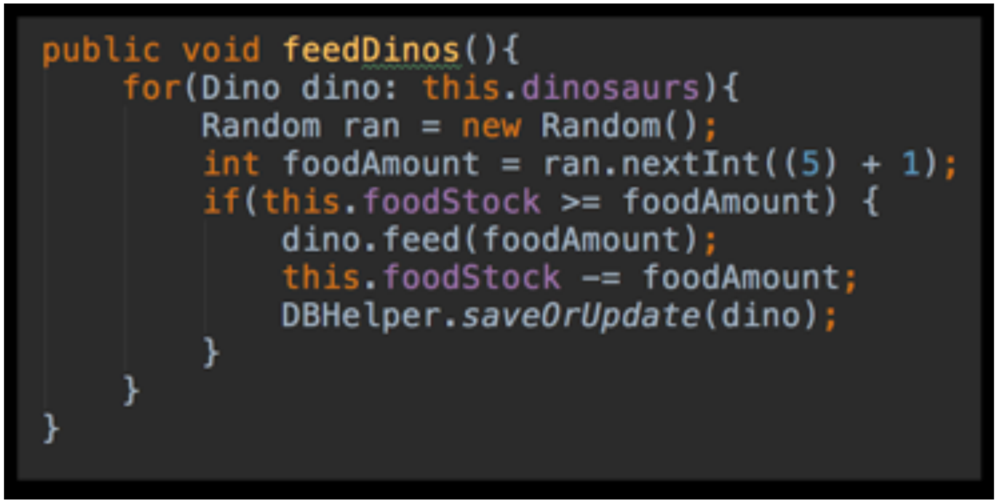


```
RecordCollector.prototype.genreValue = function (genre) {
  let filteredRecords = _.filter(this.collection, {genre: genre});
  return _.sumBy(filteredRecords, 'price');
};
```

I’d chosen to use this algorithm to calculate the total value of a record collectors collection of a particular genre.

Using Lodashes filter method the algorithm filters out the collection array and creates a new filtered array of records of the same genre.

The algorithm then returns the sum of all the filtered records prices using Lodashes sumBy method.

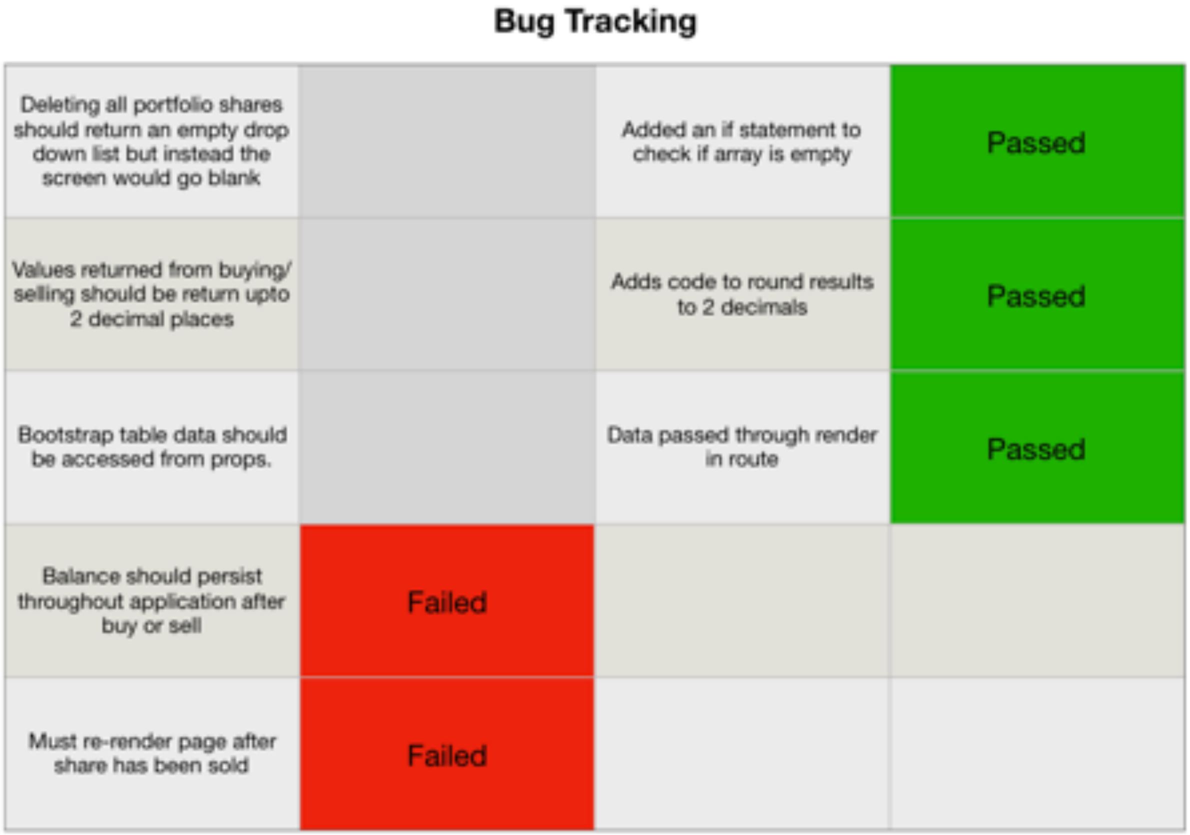


```
public void feedDinos(){
    for(Dino dino: this.dinosaurs){
        Random ran = new Random();
        int foodAmount = ran.nextInt((5) + 1);
        if(this.foodStock >= foodAmount) {
            dino.feed(foodAmount);
            this.foodStock -= foodAmount;
            DBHelper.saveOrUpdate(dino);
        }
    }
}
```

This algorithm was chosen to feed dinosaurs contained in a particular paddocks dinosaurs array.

The algorithm loops through the paddocks array of dinosaurs and then generates a random integer value between 1 and 5 which is set to the variable foodAmount.

It then checks if the foodAmount value is greater than the paddocks foodStock value, which is set at 0 initially, and if true the dinosaur will feed from the paddocks foodStock. The foodStock value will then be subtracted from the variable foodAmount, and then update the dinosaurs stomach value in the database.

	P	P 17	Produce a bug tracking report	
				

Bug Tracking

Deleting all portfolio shares should return an empty drop down list but instead the screen would go blank		Added an if statement to check if array is empty	Passed
Values returned from buying/ selling should be return upto 2 decimal places		Adds code to round results to 2 decimals	Passed
Bootstrap table data should be accessed from props.		Data passed through render in route	Passed
Balance should persist throughout application after buy or sell	Failed		
Must re-render page after share has been sold	Failed		