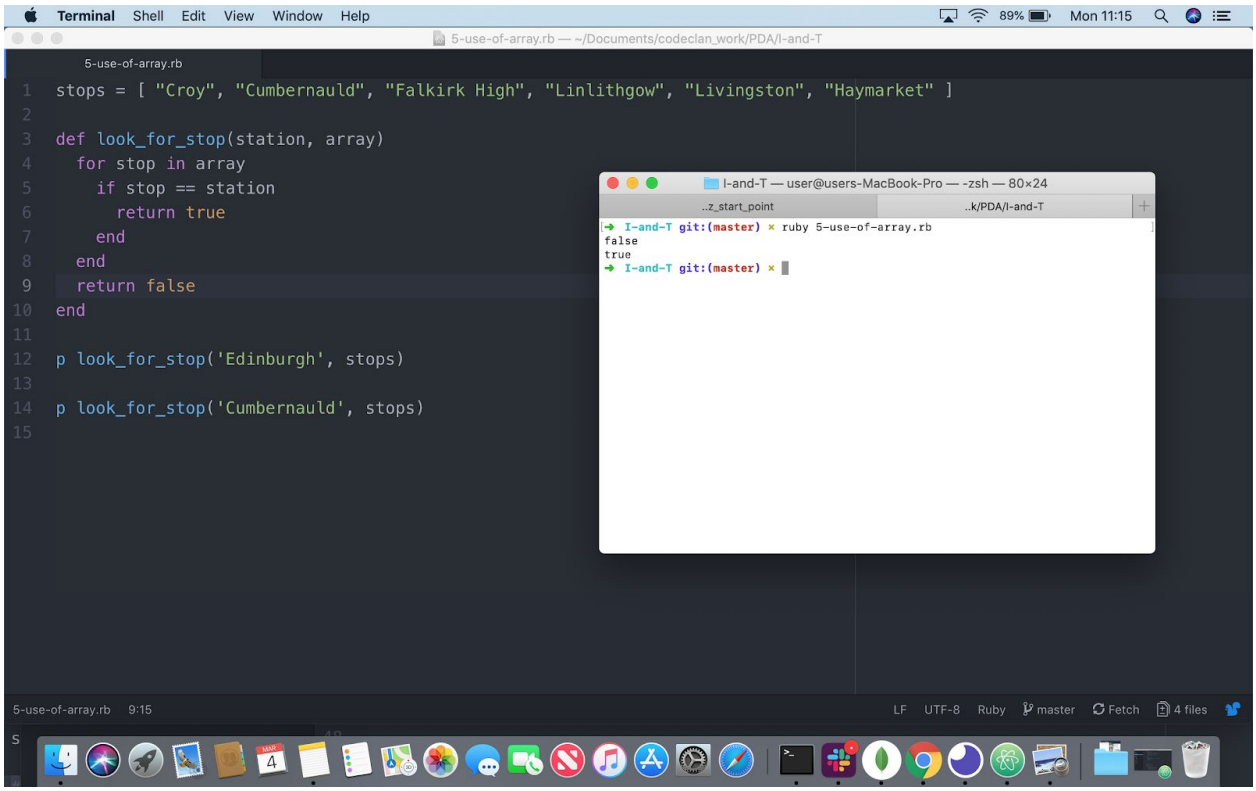
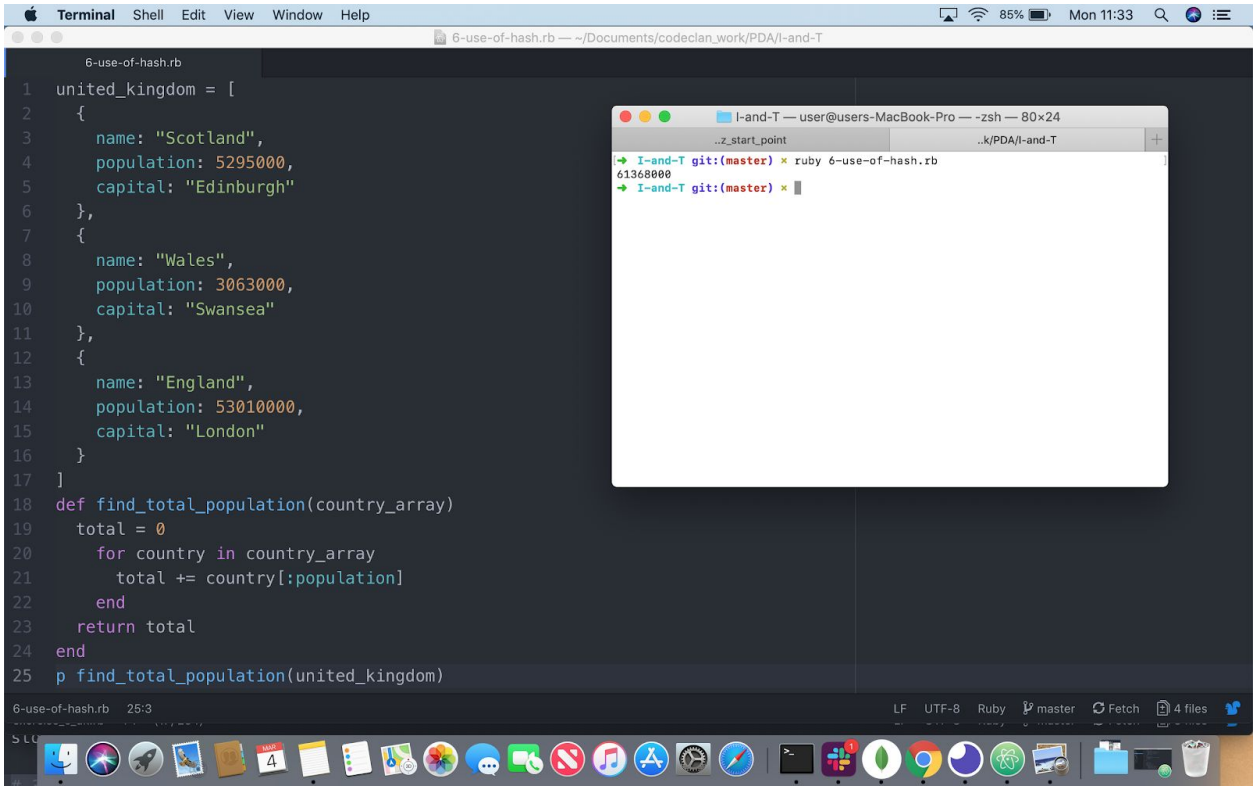
PDA Evidence Portfolio - Cassia Lonsdale

I.T.5. Use of an array:



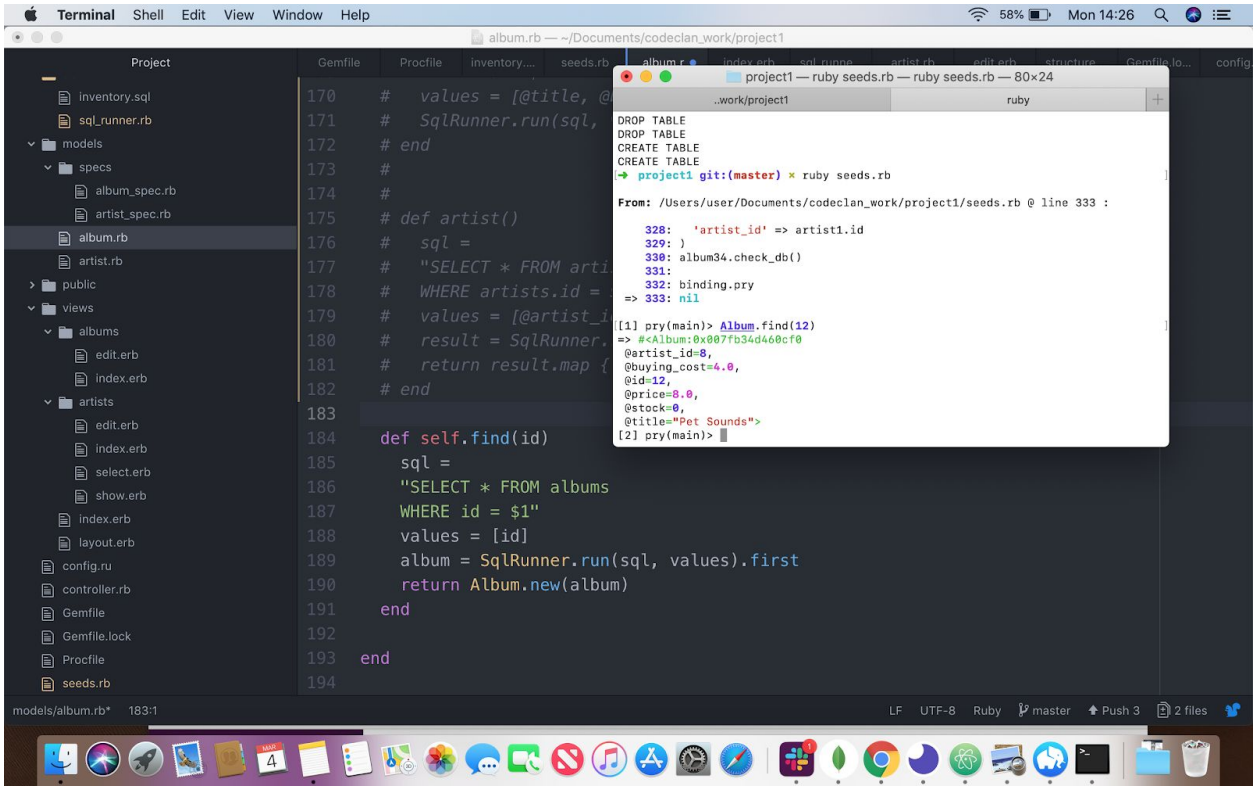
- Line 1 shows an array of strings (assigned to the variable name ‘stops’).
- Lines 3-10 show a function written for the purpose of determining whether a given station exists within the array ‘stops’.
- It goes through each item in the array and if that item matches the station being searched for (passed into the function as a parameter on line 3), the function returns true.
- If it goes through the entire array and does not find the given station, it returns false.

I.T.6. Use of a hash:



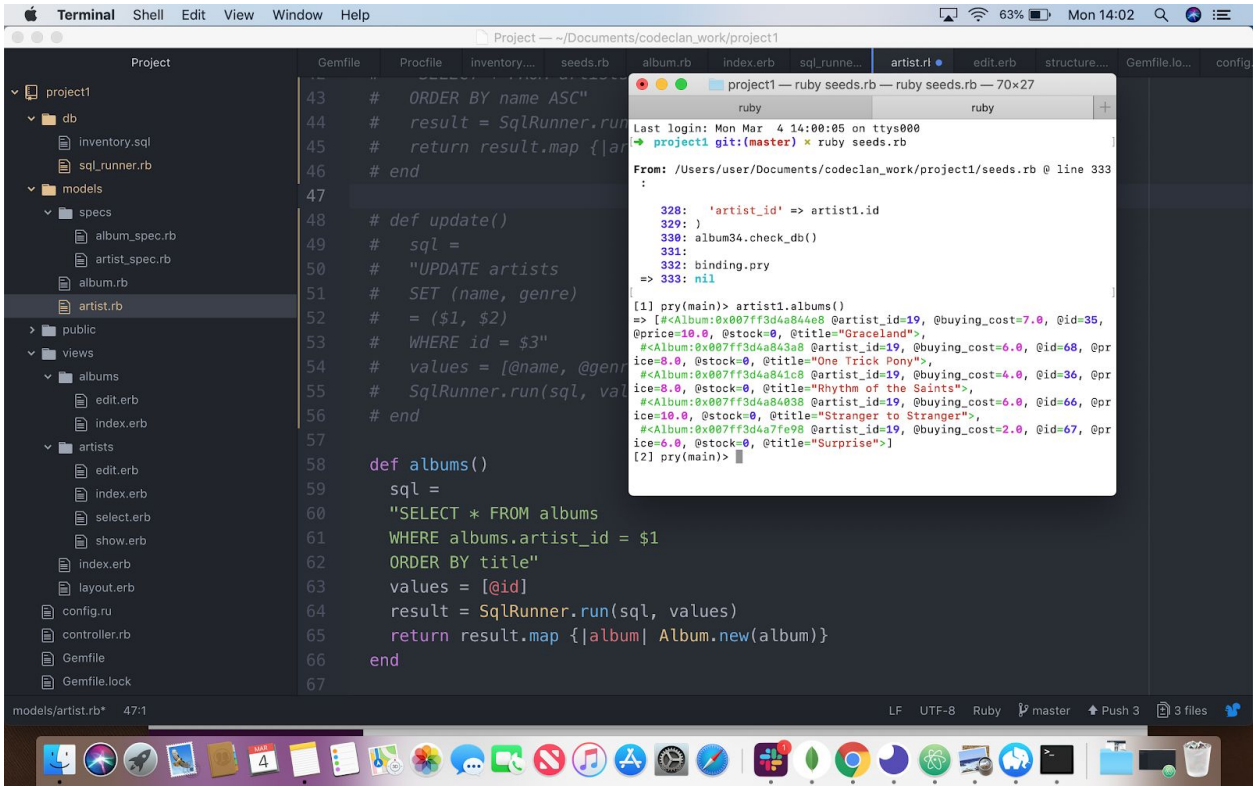
- Lines 1-17 show an array of hashes, each hash showing a country with 3 keys (name, population and capital).
- Lines 18-24 show a function written for the purpose of calculating the total population of countries in a given array of hashes.
- It starts by setting total to 0, then it goes through each item in the array and accesses the population by the ‘population’ key, each time adding this number to the total.
- Once it has gone through each item it returns the total.

I.T.3. Searching for data:



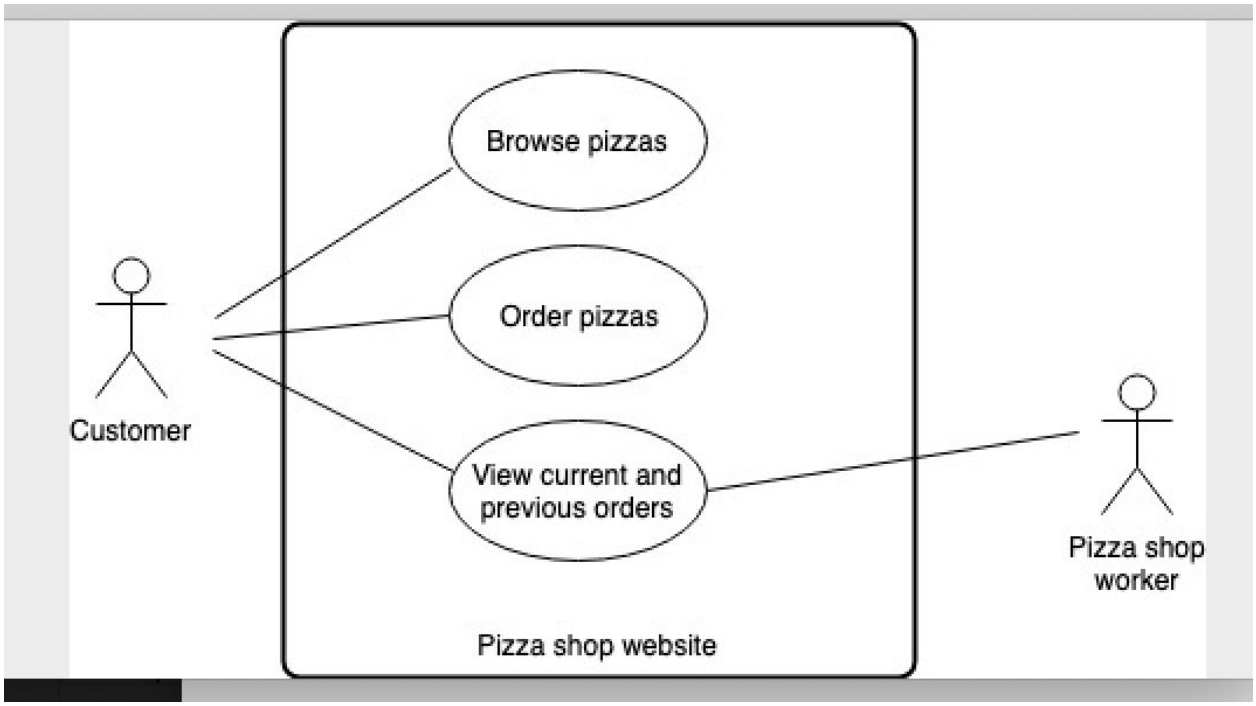
- The code above shows the function I wrote for my project that was used to find a particular album from a database using only its ID, which it takes in as a parameter (line 184). The function is called on the Album class.
- I was writing in Ruby and inserted SQL to run my search (lines 186-7).
- Line 189 calls a helper method I wrote in a separate document to process the SQL correctly.
- As you can see from the console, my function returned the expected album (the album object with id=12).

I.T.4. Sorting data:



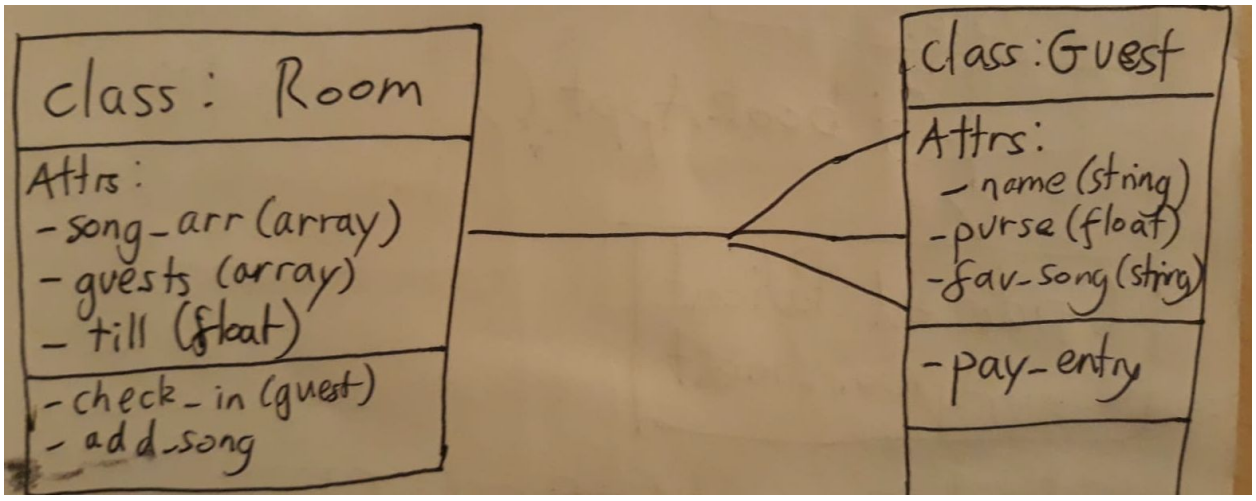
- The code above shows the function I wrote for my project that was used to find all the albums in a database that are listed under a particular artist. The function is called on an Artist object.
- The artist is identified by the artist's id, which I joined to the album table via the column 'artist_id'.
- As stated above, I was writing in Ruby and inserted SQL to run my search (lines 60-62). Line 62 sorts the selected data by the 'title' column, returning the data in alphabetical order.
- Line 64 calls the same helper method I referred to in I.T.3.
- As you can see from the console, my function returns all albums by Paul Simon.

A.D.1 Use case diagram:



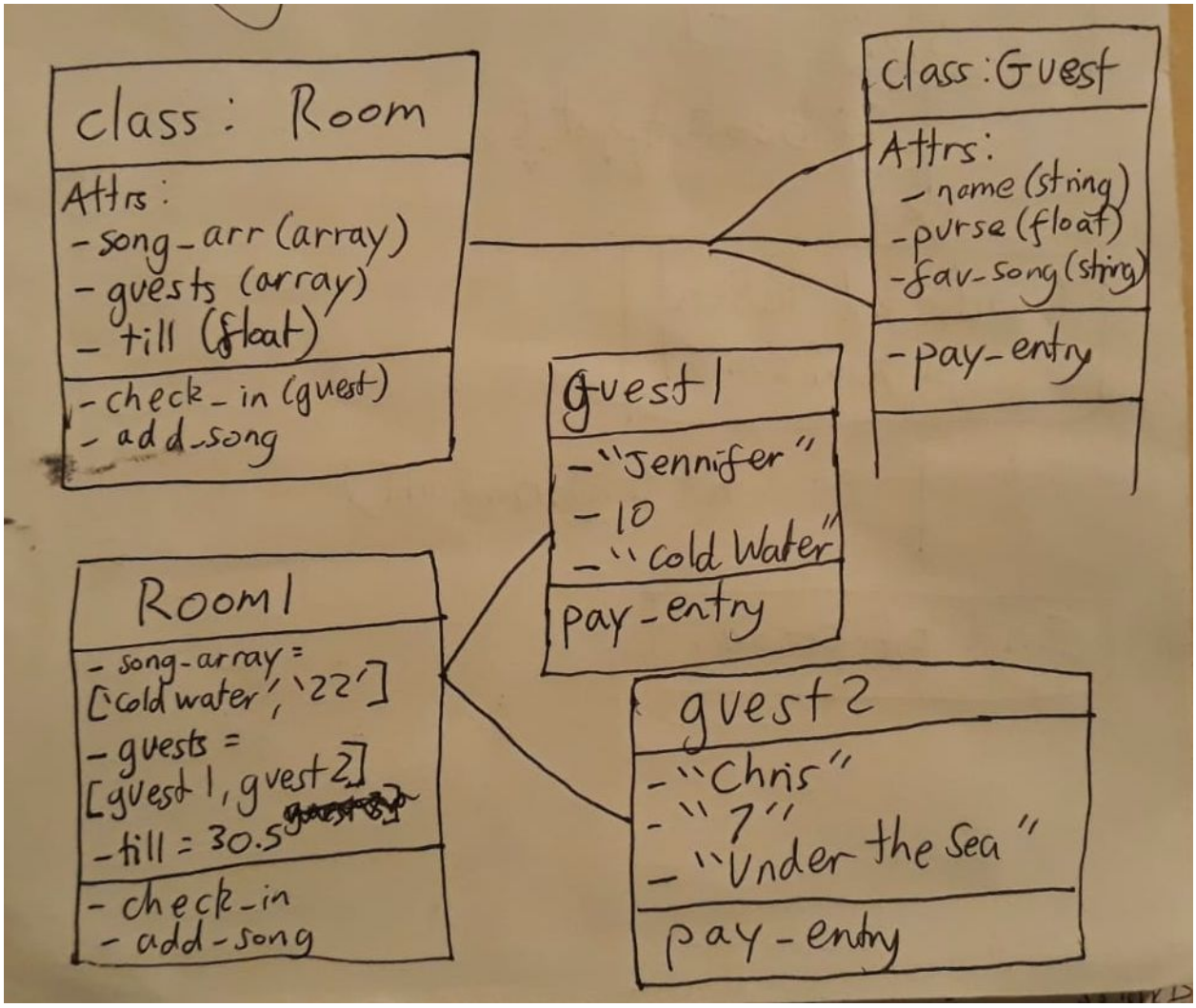
Above is a use case diagram I drew to help me visualise how a pizza shop website might be used by both customers and staff to process pizza orders. A customer can browse the menu and place orders, as well as review their order and previous orders they have made. The staff can obviously also view current orders, as well as previous orders (possibly for marketing purposes).

A.D.2. Class diagram:



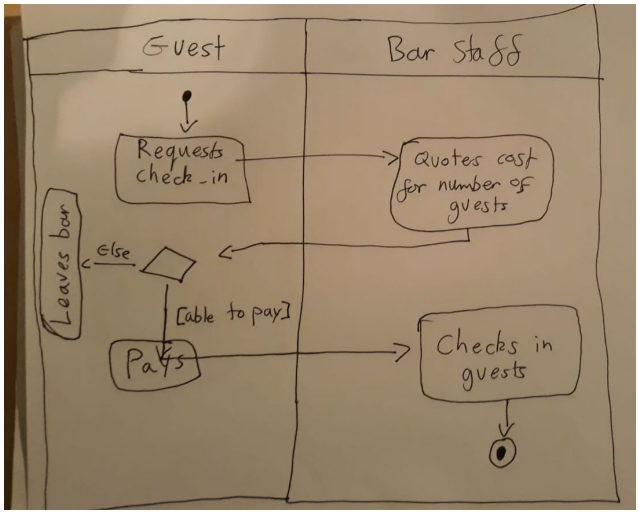
Above is a photo of class diagrams I drew when designing a karaoke bar check-in system. As the diagram shows, it is a one to many relationship (one room in the bar has many guests). The Room class also has a method to add a guest to its guest array.

A.D.3. Object diagram:



Above are object diagrams I drew detailing instances of the Room and Guest classes illustrated above. The diagrams give the actual values of the attributes defined in the class diagrams. The room has both guests in its guest array.

A.D.4. Activity diagram:



Above is an activity diagram I drew to help me visualise the sequence of activities a guest and bar staff would carry out in order to check a guest into the bar. The activities go back and forth between guest and staff. After the guest requests a check-in, depending on whether the guest has the amount of money specified by the bar staff, the guest either leaves the bar or pays the specified money, and is then checked in by the staff.

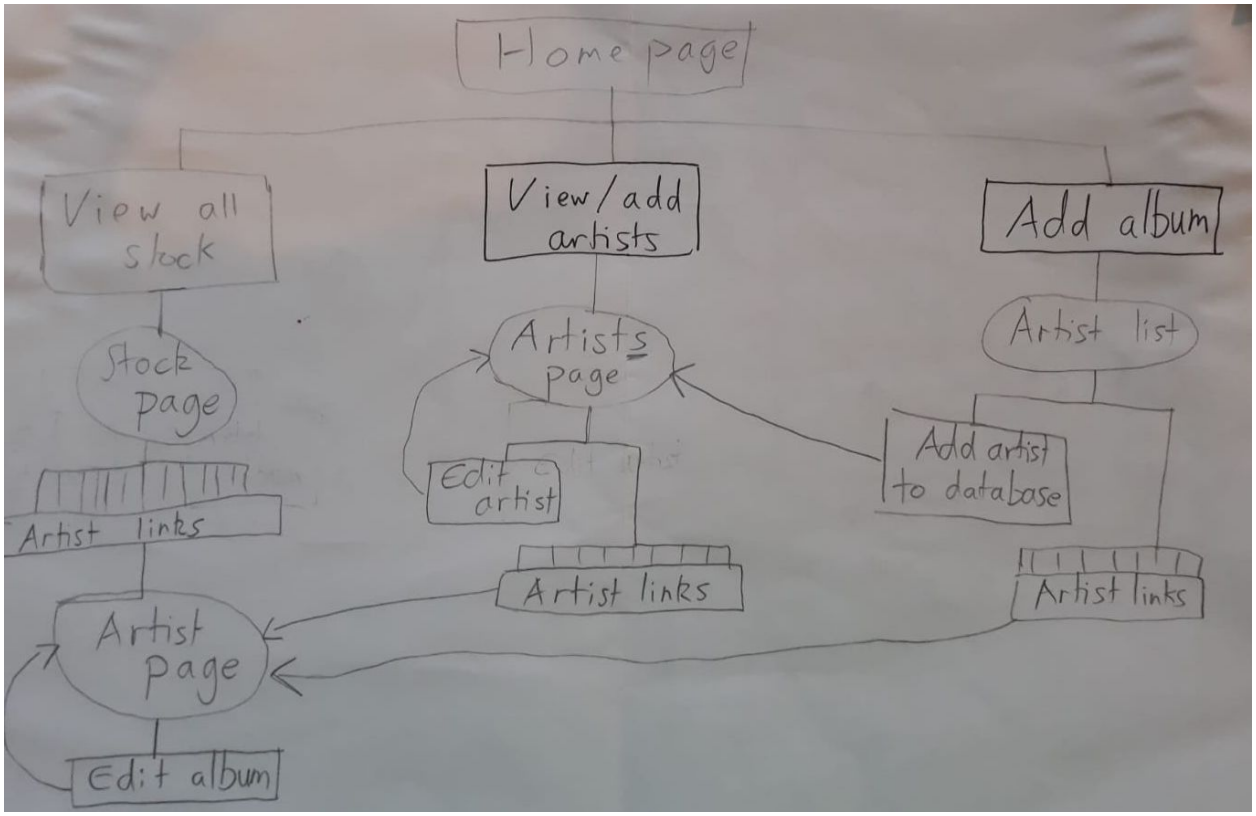
A.D.6. Implementations constraints plan:

Constraint category	**Implementation constraint**	**Solution**
Hardware and software platforms	For this project I was only allowed to use Ruby, which meant certain dynamic features were unavailable to me. I wanted to include a drop-down list of artists for the user to select from when adding an album to the database, but this was not possible.	I had to re-plan my interface, and instead list all my artists statically on the page and make each artist a link to another page. This alternative interface was clear and usable enough for my purposes.
Performance requirements	The intended purpose of the app is for store managers/staff to be able to update stock as and when it is bought/sold, so efficiency/speed is important.	I ensured only necessary data was stored in the database. Data that could be calculated based on existing data was not included in the back-end, so as to not use unnecessary storage.
Persistent storage and transactions	My program had to allow shop staff to keep closing the program and coming back to it as it was designed to be long-term storage for data.	I ensured my data was stored in a database in a separate back-end with a router to retrieve and save information, so that data persisted after the program was closed.

Usability	As already stated, the intended purpose of the app is for store managers/staff to be able to update stock as and when it is bought/sold, which means it has to be easy to navigate and quick to learn how to use.	There are no unnecessary frills in the UI, and it is easy to navigate back and forth between tables and the homepage, with obvious and clear links.
Budgets	I had no money to complete this project. I was therefore unable to hire staff to complete it for me.	I overcame this constraint by doing all the work myself.
Time limitations	The program had to be designed and created within one week.	I dealt with this strict time constraint by constantly being aware that I had to prioritise which features to ensure were complete and working, and decide which ones were not part of the MVP so had to be abandoned.

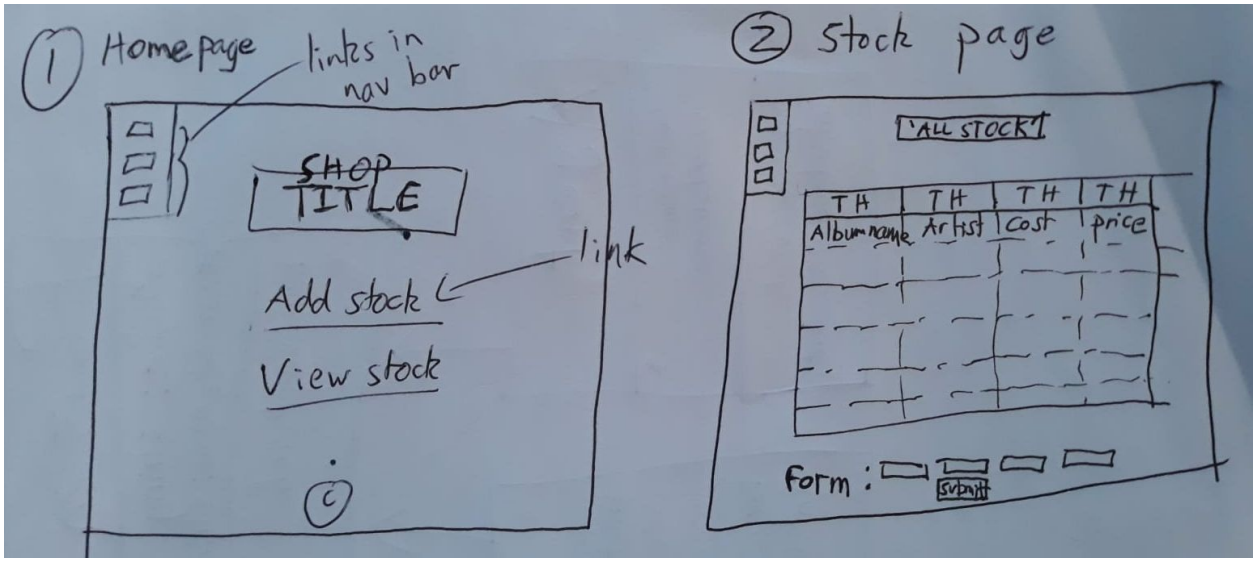
Above is an implementations constraints plan I wrote when planning my individual project (building a program to help shopkeepers keep track of their stock). I had to take into account what constraints or problems might be posed by the factors listed in the first column. The second column summarises the nature of the constraint or difficulty I predicted might be posed by that category. The third column summarises how I decided to overcome that particular constraint or difficulty.

P.5. User site map:



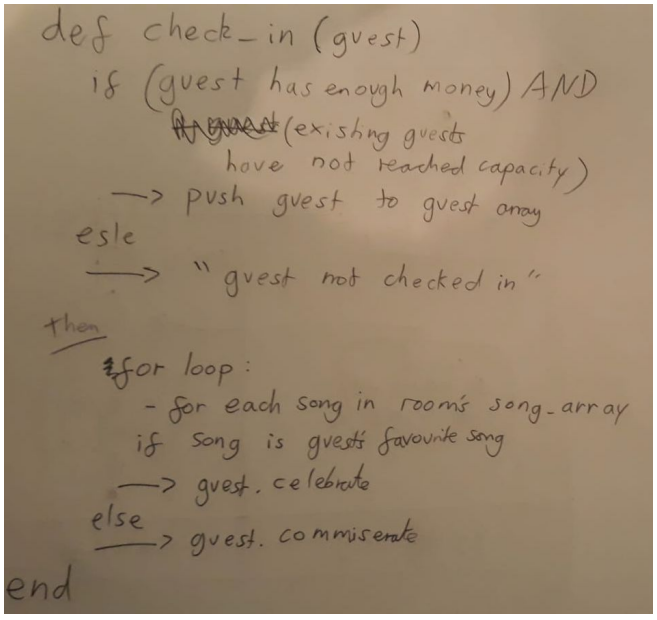
Above is a user site map for my shop inventory program. It is laid out like a tree with the Homepage at the top, and the three links the user can access from there are listed underneath as three different branches. The branches continue to specify what further pages are linked to from the current page. Some branches end up in the same place as other branches. For example, when a user goes down the ‘Add album’ branch, they may need to add the relevant artist to the database, which redirects them to the Artists page (as this contains the form needed to add the new artist).

P.6. Wireframe diagrams:



Above are two draft wireframes I drew when planning my shop inventory project. They are rough outlines of how I envisaged my Homepage and Stock page to look. The pages ended up looking similar to this, except the navbar I ended up moving to the top centre of the page, and the ‘Add stock’ link on the Homepage turned into two links (one for artist, one for album).

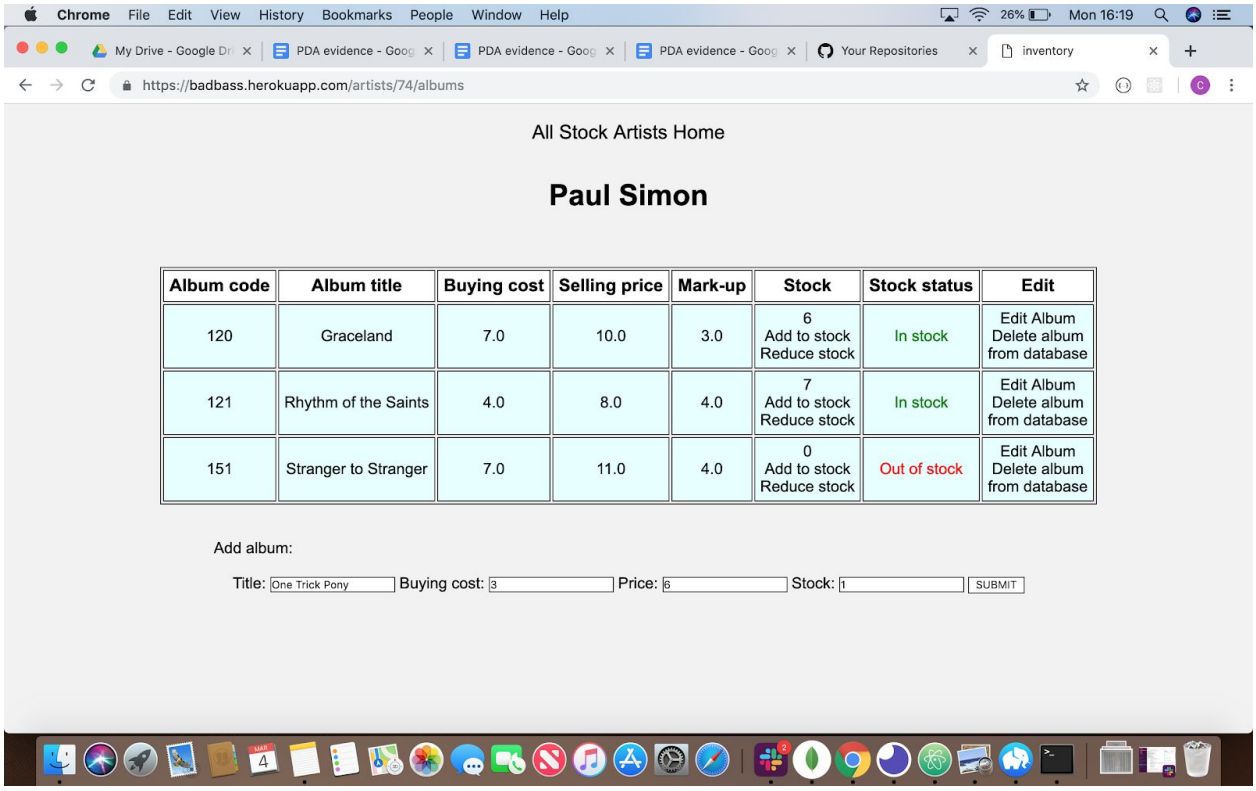
P.10. Pseudocode:

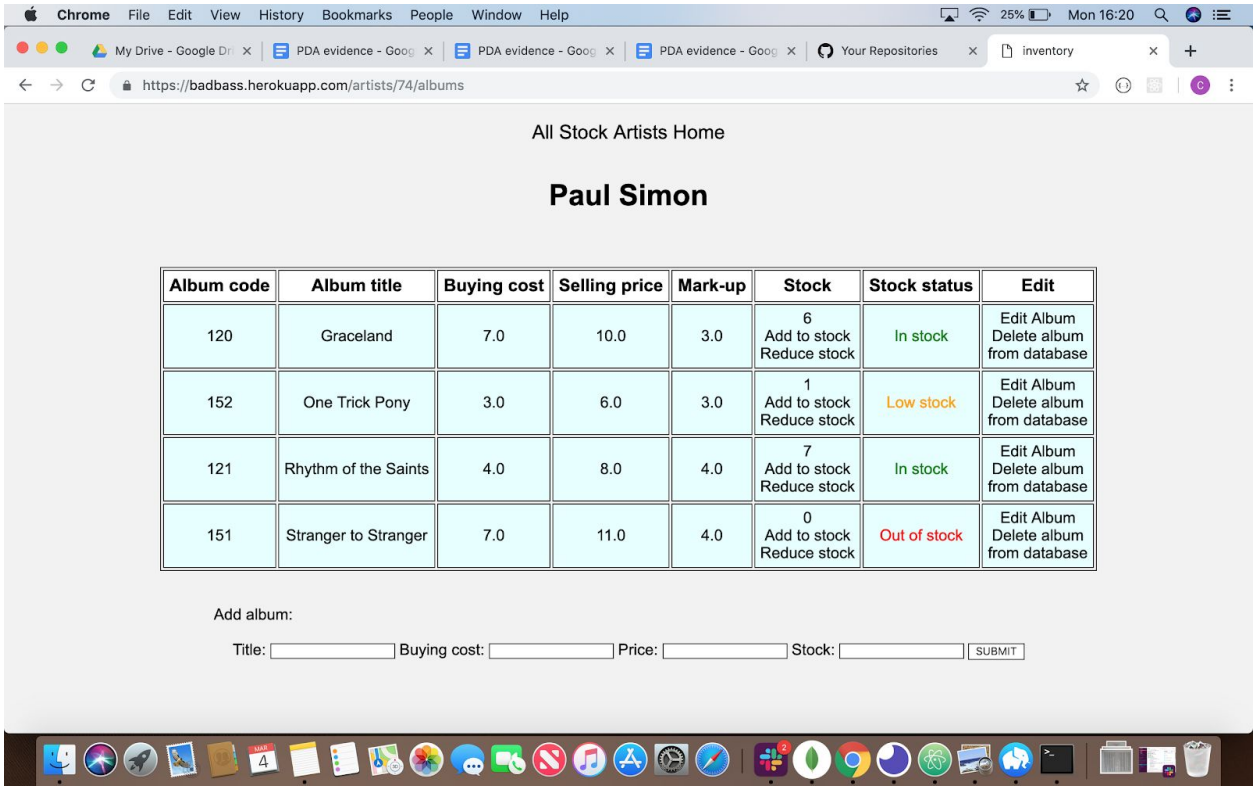


Above is some pseudocode I drafted when trying to work through in my head how my check-in function would work for my karaoke bar system, and the various things it had to check when a guest requests a check-in. The code does not use Ruby syntax, but instead outlines the logic of

how the function would work, before the minutiae was worked out and the function properly written.

P.13. User input being processed:





The above screenshots show the Paul Simon album page from my project. The first screenshot shows the user having inputted details of a new album they wish to add to the database. The second screenshot shows the new rendering of the page after the form had been submitted. These details have been processed and are now included in the list of albums.

P.14. Interaction with data persistence:

localhost › inventory › artists Connected. PostgreSQL 10.5 LOCAL

id	name	genre
14	Abba	Pop
6	Backstreet Boys	Pop
7	Bob Marley	Other
3	Cosmo Jarvis	Other
11	Ella Fitzgerald	Jazz
4	Fleetwood Mac	Other
10	Frank Sinatra	Jazz
16	Maximum the Hormone	Rock
12	Muse	Rock
2	Paul Simon	Other
15	Porcupine Tree	Other
5	Taylor Swift	Pop
8	The Beach Boys	Other
17	The Proclaimers	Other
13	The Rolling Stones	Rock
1	Various	Other
9	Weird Al Yankovic	Other

Content | Structure | DDL | + Row 17 rows Filter Page 1 of 1

Chrome File Edit View History Bookmarks People Window Help 22% Mon 16:33

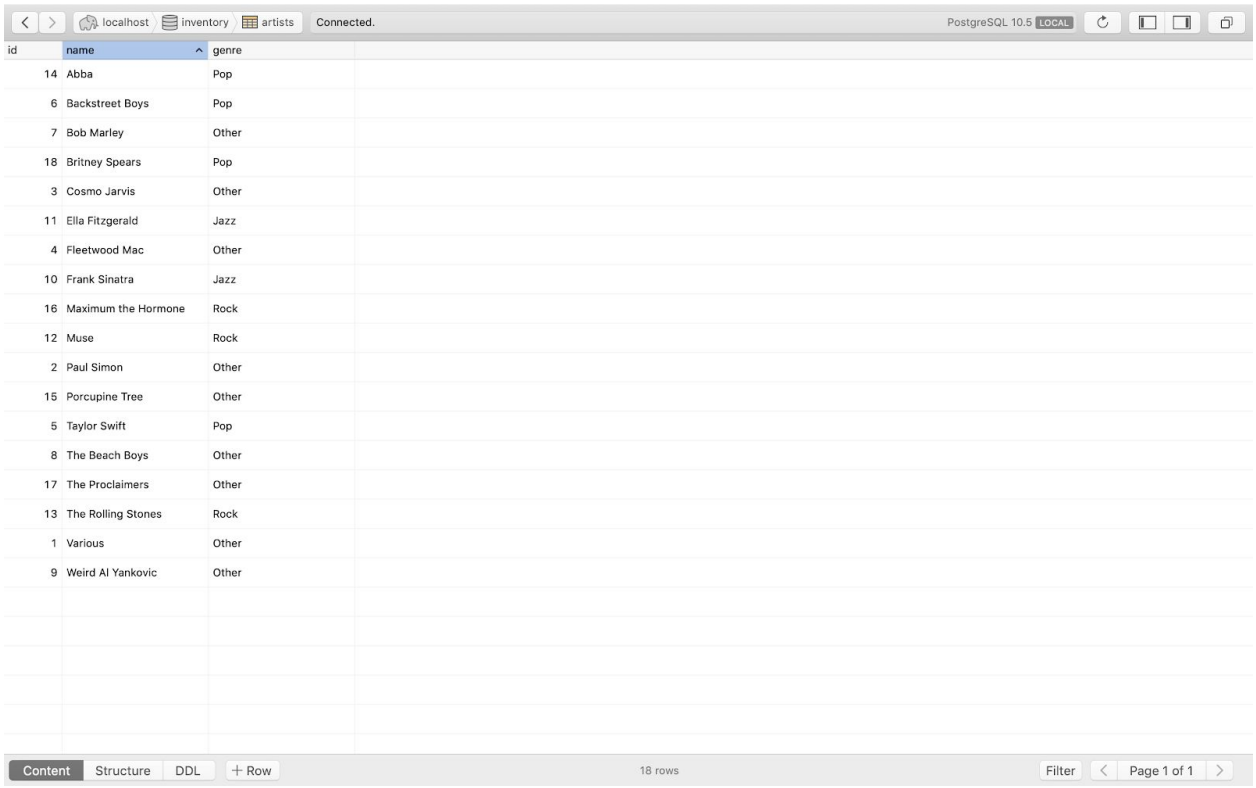
My Drive - Google Dr | PDA evidence - Goog | PDA evidence - Goog | PDA evidence - Goog | Your Repositories | inventory

localhost:4567/artists

			from database	
10	Frank Sinatra	Jazz	Delete artist from database	Edit artist
16	Maximum the Hormone	Rock	Delete artist from database	Edit artist
12	Muse	Rock	Delete artist from database	Edit artist
2	Paul Simon	Other	Delete artist from database	Edit artist
15	Porcupine Tree	Other	Delete artist from database	Edit artist
5	Taylor Swift	Pop	Delete artist from database	Edit artist
8	The Beach Boys	Other	Delete artist from database	Edit artist
17	The Proclaimers	Other	Delete artist from database	Edit artist
13	The Rolling Stones	Rock	Delete artist from database	Edit artist
1	Various	Other	Delete artist from database	Edit artist
9	Weird Al Yankovic	Other	Delete artist from database	Edit artist

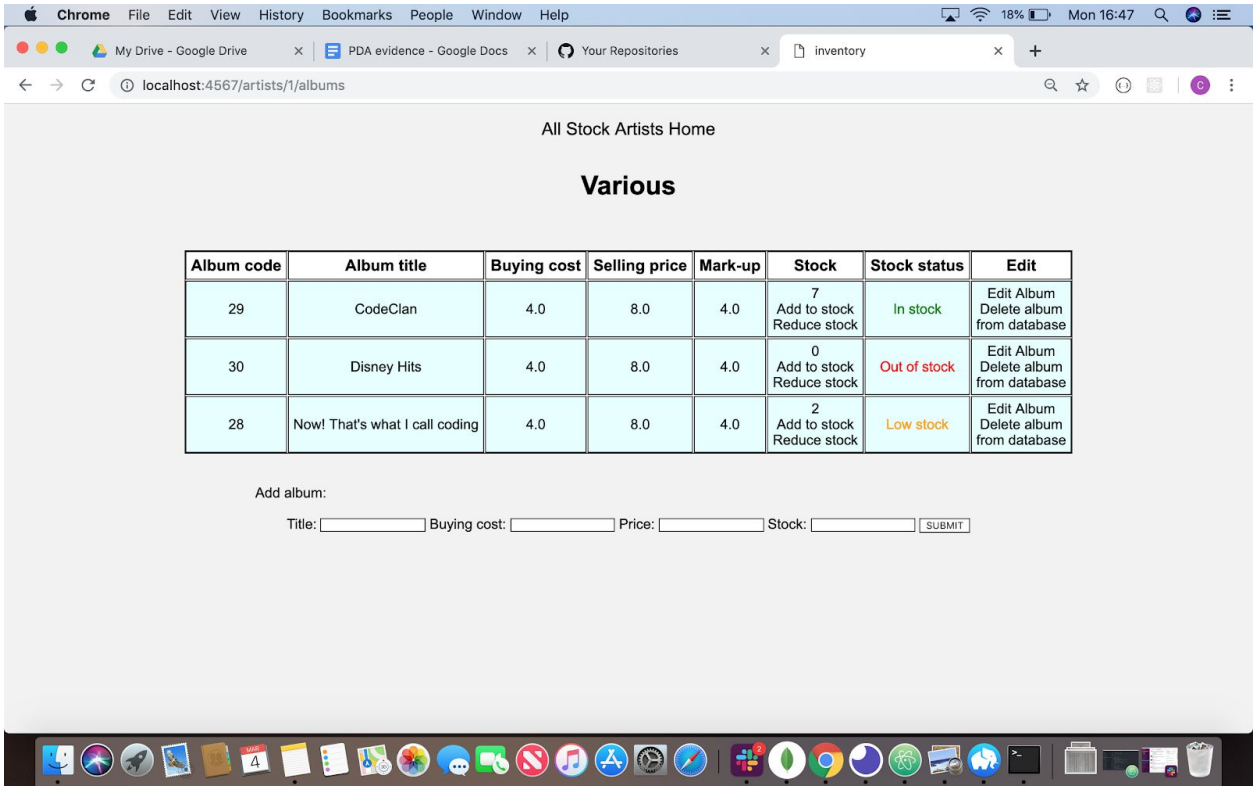
Add artist:

Name: Britney Spears Genre: Pop SUBMIT



The above screenshots show the artists in my database, both from the back-end and front-end. The first screenshot shows the artists listed in my database (on Postico). The second screenshot shows the user having inputted details of a new artist that they wish to add to the database, via the front-end artists page form. The third screenshot shows the new artist list in Postico after the form had been submitted. These form details have been processed and have persisted in the back-end.

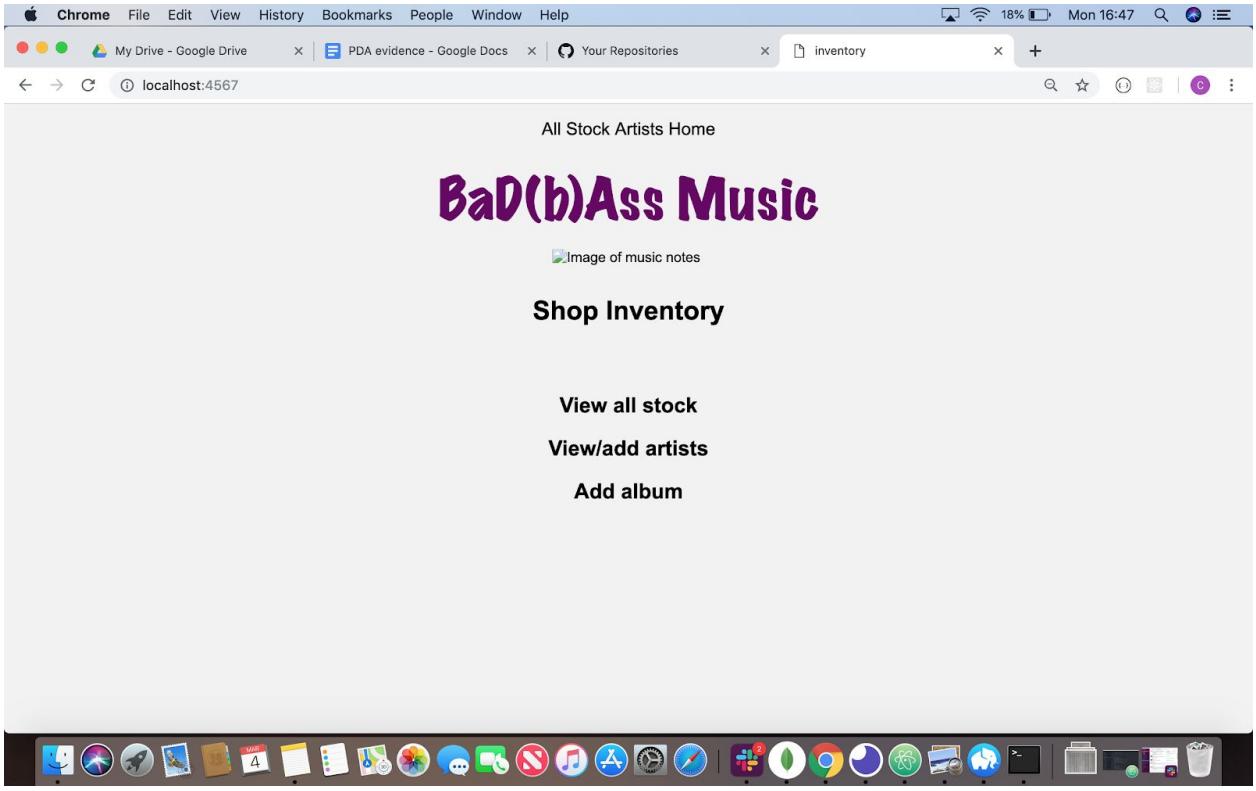
P.15. Output of results and feedback to user:



The above screenshots demonstrate my program providing expected feedback to the user. The first screenshot shows my table of all stock, and the user clicking on a link to the artist belonging

to a particular album. The second screenshot shows what is returned to the user after clicking on that link: the user sees all albums listed under that artist.

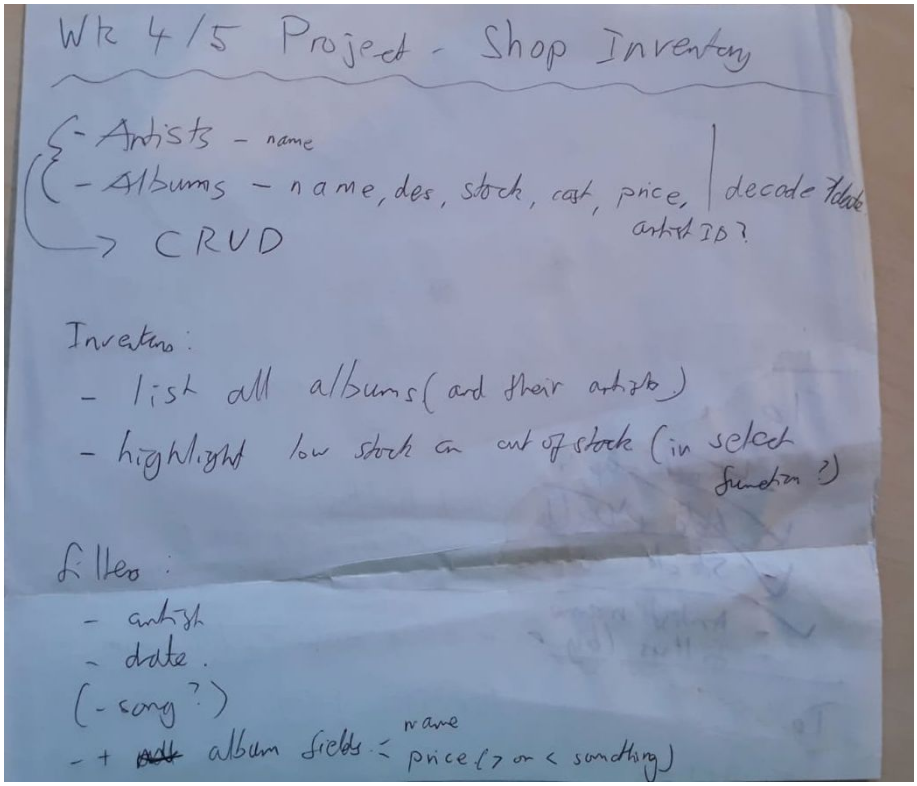
P.11. Individual project and Github repo:

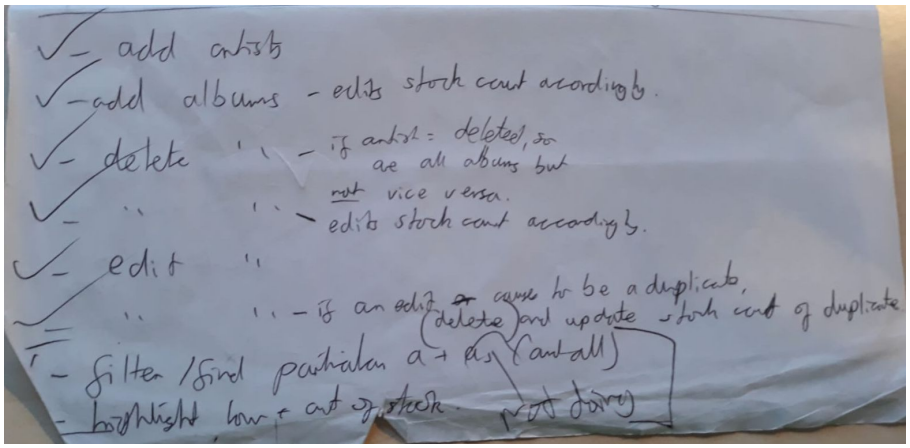


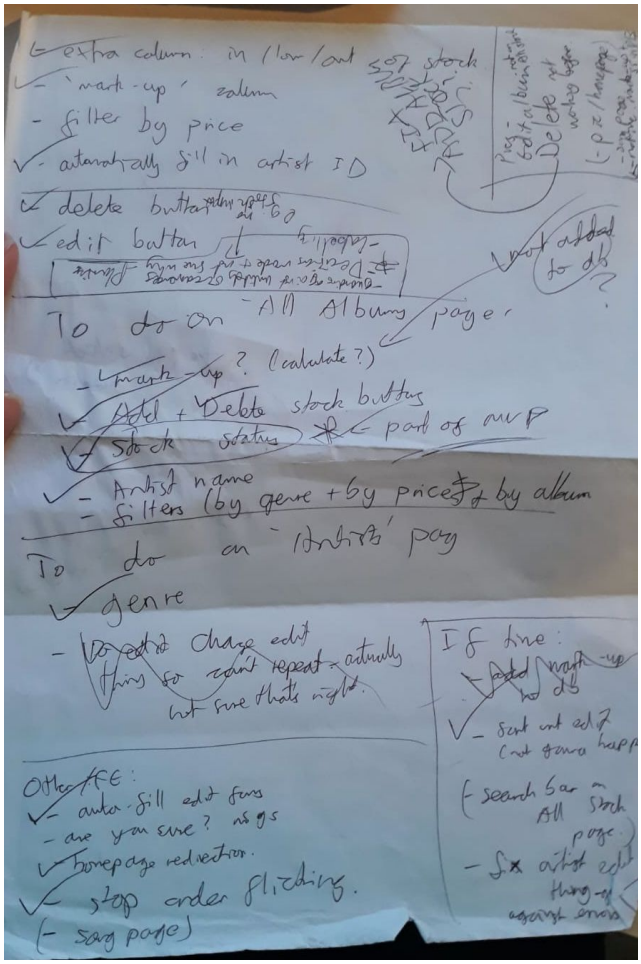
https://github.com/cass3836/Project1_shop_inventory

Above is a screenshot of the Homepage of the shop inventory tracking program I created for my individual project, displayed locally. I have also included the Github link to my repo with all the code.

P.12. Planning/stages of development:

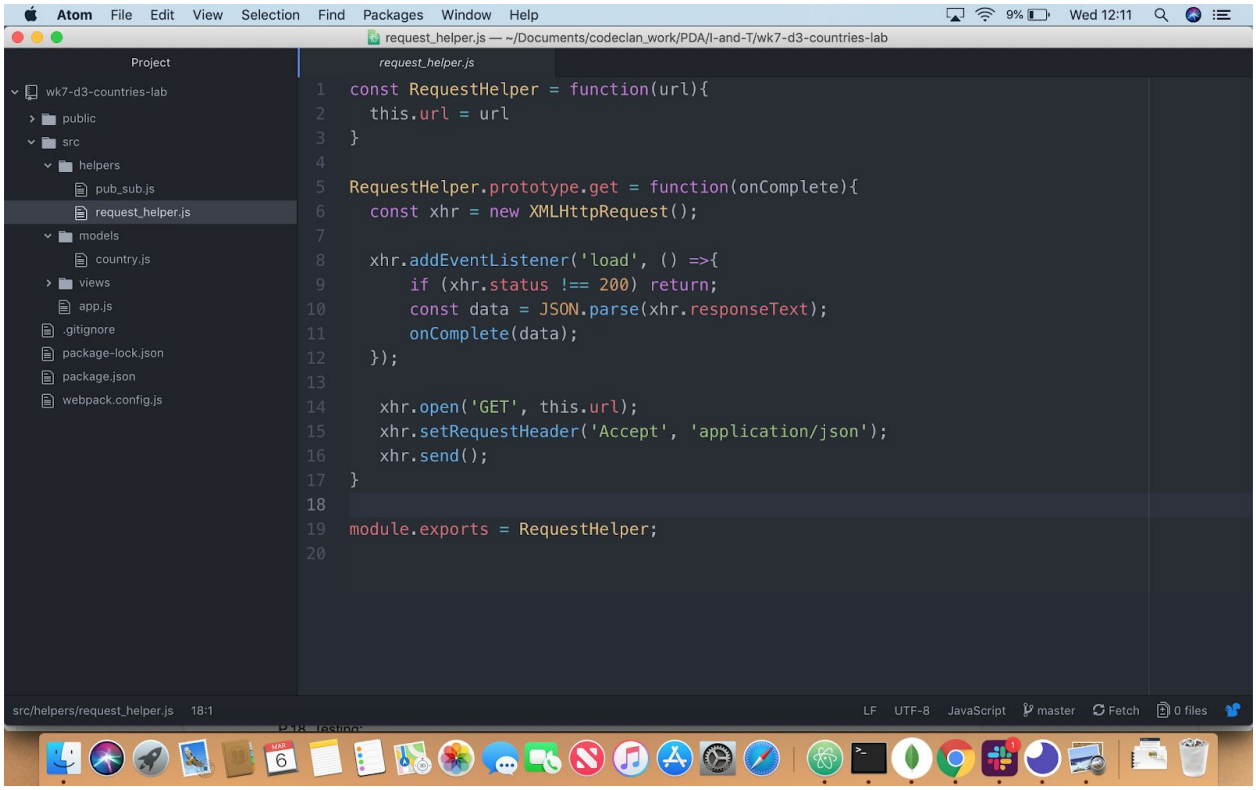

Wk 4/5 Project - Shop Inventory
- Artists - name
- Albums - name, des, stock, cost, price, | decade ?date
artist ID ?
→ CRUD
Inventory:
- list all albums (and their artists)
- highlight low stock or out of stock (in select function ?)
filter:
- artist
- date.
(- song ?)
- + ~~add~~ album fields. = name price (> or < something)

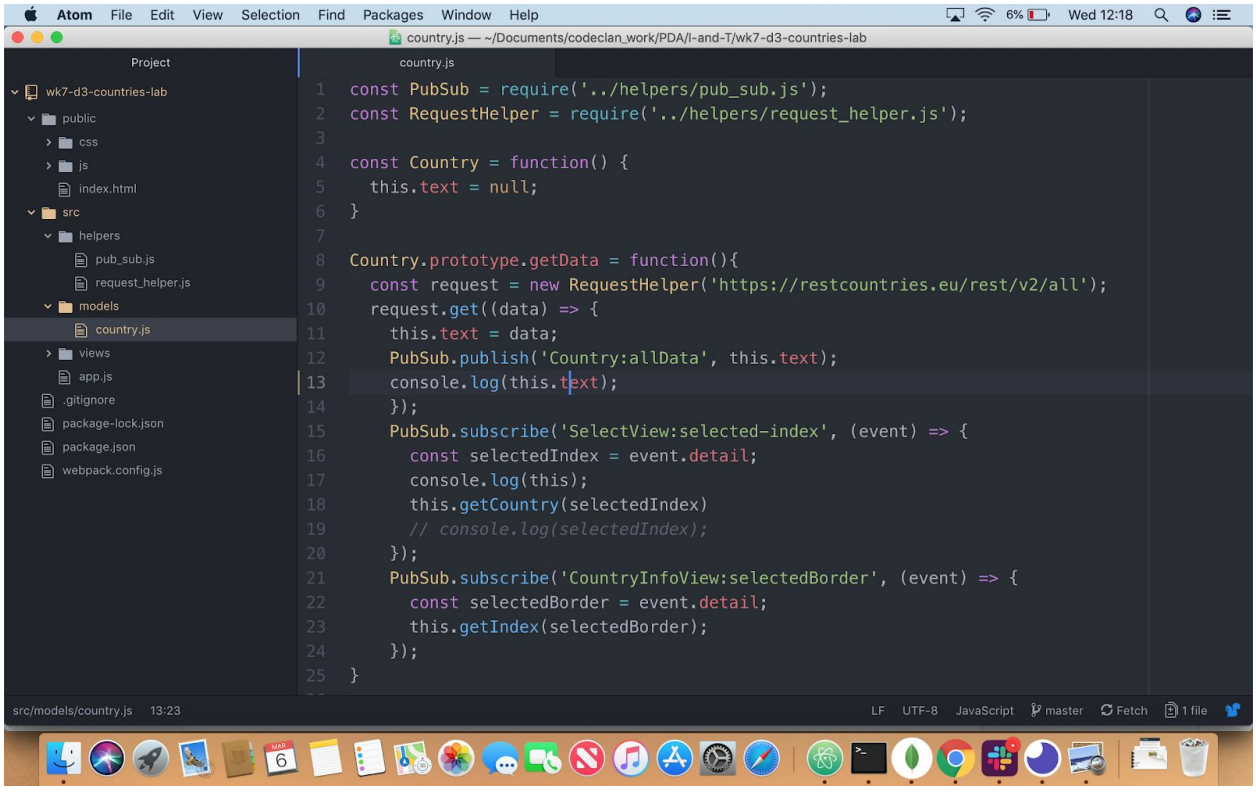

✓ - add artists
✓ - add albums - edits stock count accordingly.
✓ - delete " " - if artist = deleted, so are all albums but not vice versa.
✓ - " " " - edits stock count accordingly.
✓ - edit "
" " " - if an edit cause to be a duplicate, delete and update stock count of duplicate
- filter / find particular artists (and all)
- highlight low + out of stock. not done

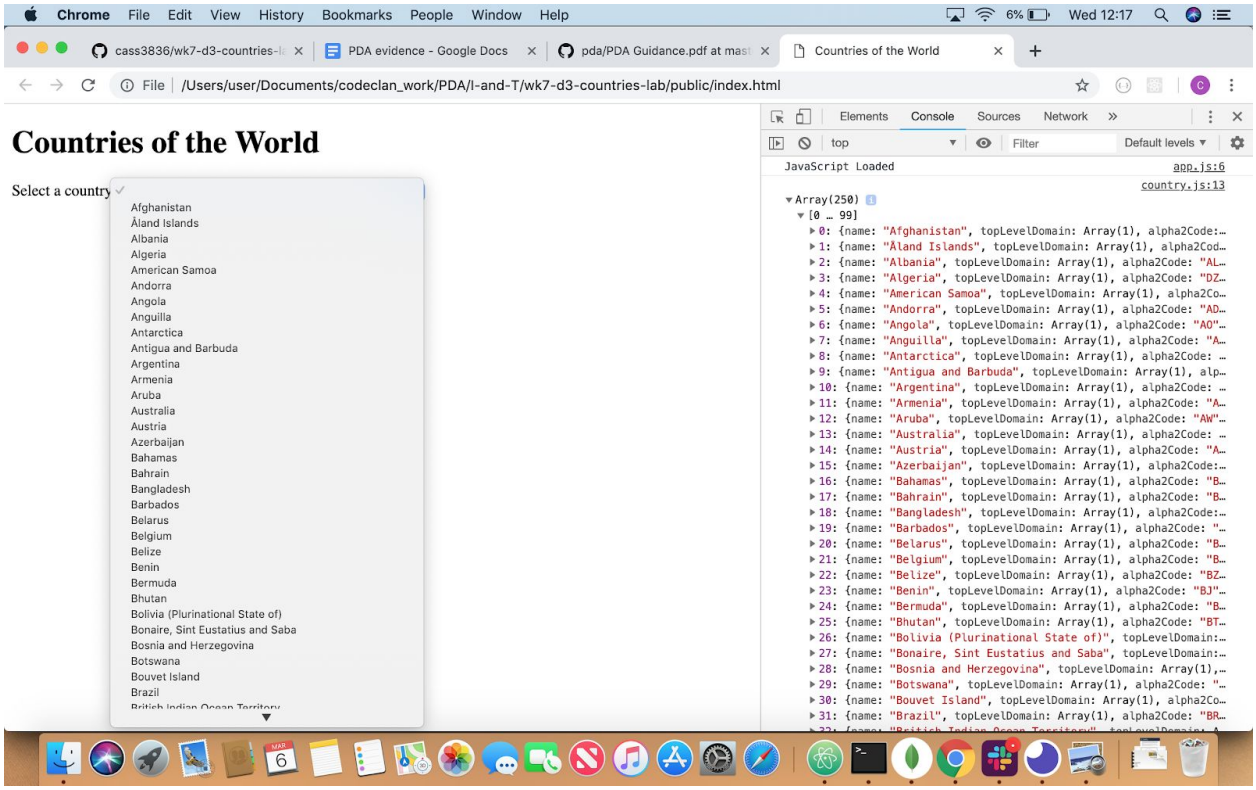


Above are photos of planning I completed before I began coding my shop inventory management system, as well as during the development process. The first photo is the very initial brainstorm of what my MVP would include. The second photo is a to do list I completed half way through my project of unfinished functionality. The third photo is a final planning session I completed, establishing what functionality I needed to add and fix, as well as extensions I was hoping to get to if I had time.

P.16. Use of an API:

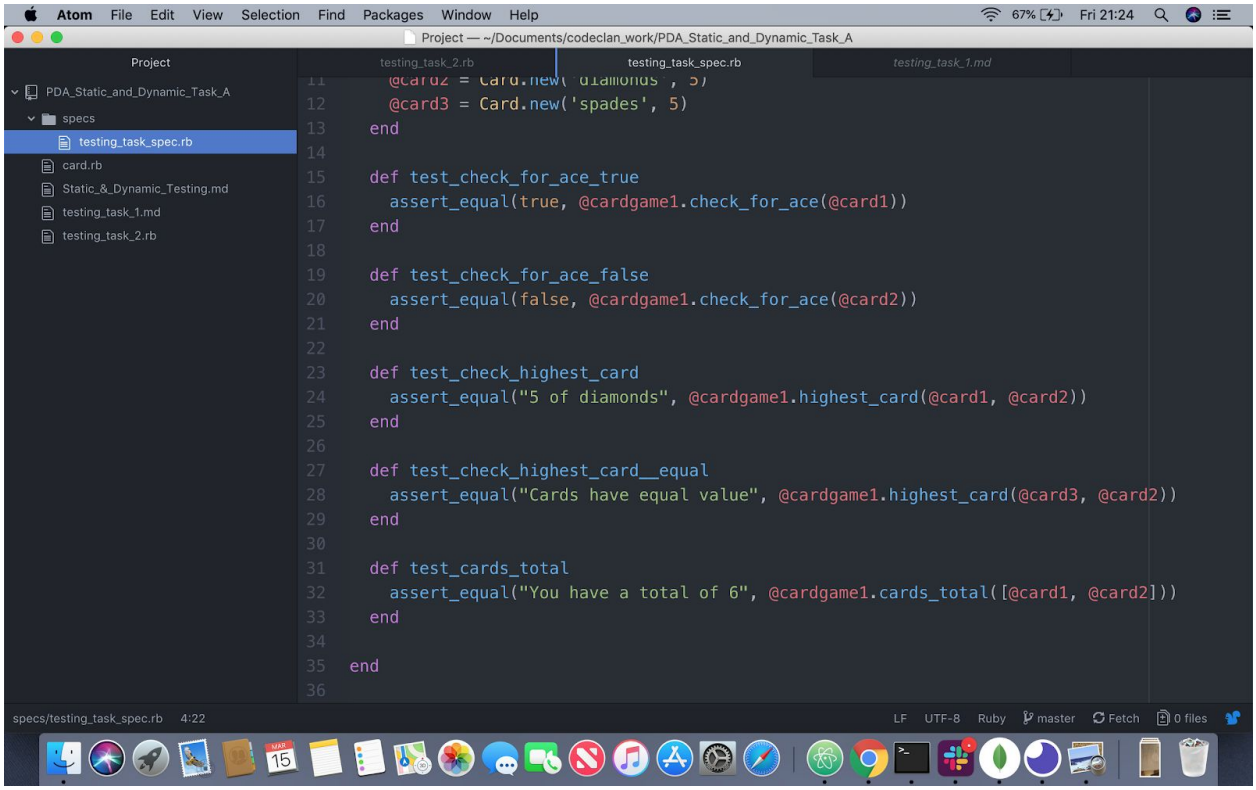

Atom File Edit View Selection Find Packages Window Help
9% Wed 12:11
request_helper.js — ~/Documents/codeclan_work/PDA/l-and-T/wk7-d3-countries-lab
Project
wk7-d3-countries-lab
public
src
helpers
pub_sub.js
request_helper.js
models
country.js
views
app.js
.gitignore
package-lock.json
package.json
webpack.config.js
request_helper.js
const RequestHelper = function(url){
this.url = url
}
RequestHelper.prototype.get = function(onComplete){
const xhr = new XMLHttpRequest();
xhr.addEventListener('load', () =>{
if (xhr.status !== 200) return;
const data = JSON.parse(xhr.responseText);
onComplete(data);
});
xhr.open('GET', this.url);
xhr.setRequestHeader('Accept', 'application/json');
xhr.send();
}
module.exports = RequestHelper;
src/helpers/request_helper.js 18:1
LF UTF-8 JavaScript master Fetch 0 files


Atom File Edit View Selection Find Packages Window Help
6% Wed 12:18
country.js — ~/Documents/codeclan_work/PDA/l-and-T/wk7-d3-countries-lab
Project
wk7-d3-countries-lab
public
css
js
index.html
src
helpers
pub_sub.js
request_helper.js
models
country.js
views
app.js
.gitignore
package-lock.json
package.json
webpack.config.js
country.js
const PubSub = require('../helpers/pub_sub.js');
const RequestHelper = require('../helpers/request_helper.js');
const Country = function() {
this.text = null;
}
Country.prototype.getData = function(){
const request = new RequestHelper('https://restcountries.eu/rest/v2/all');
request.get((data) => {
this.text = data;
PubSub.publish('Country:allData', this.text);
console.log(this.text);
});
PubSub.subscribe('SelectView:selected-index', (event) => {
const selectedIndex = event.detail;
console.log(this);
this.getCountry(selectedIndex)
// console.log(selectedIndex);
});
PubSub.subscribe('CountryInfoView:selectedBorder', (event) => {
const selectedBorder = event.detail;
this.getIndex(selectedBorder);
});
}
src/models/country.js 13:23
LF UTF-8 JavaScript master Fetch 1 file



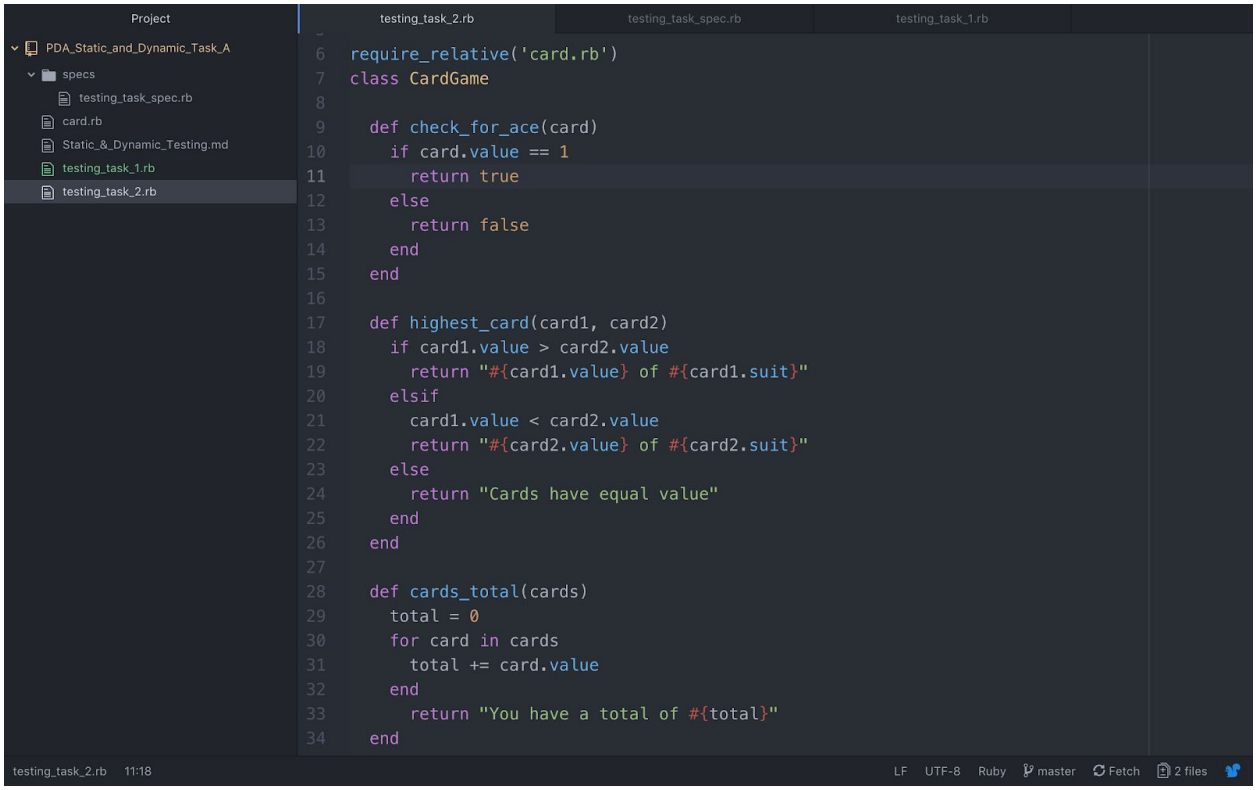
- The above screenshots show my use of a public API containing country data.
- The first screenshot shows the function I wrote to retrieve the API data. It is written as a helper method in a separate document to make the code more readable.
- The second screenshot shows this function being used in the ‘getData’ function that I wrote for a Country model. In line 9, the URL for the API is given as an argument to the RequestHelper.
- The console log on line 13 is shown in the third screenshot, where all the data from the API has been successfully retrieved and logged. (The drop-down list also shows the API being used and displayed in the UI.)

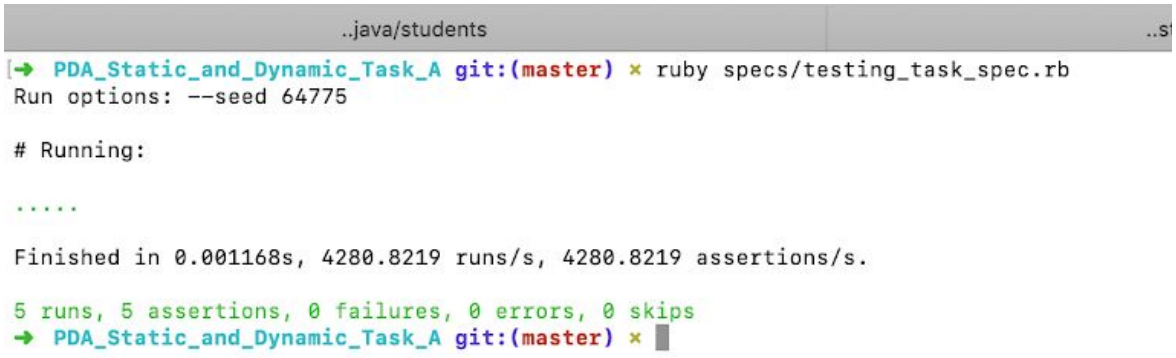
P.18. Testing:



..java/students

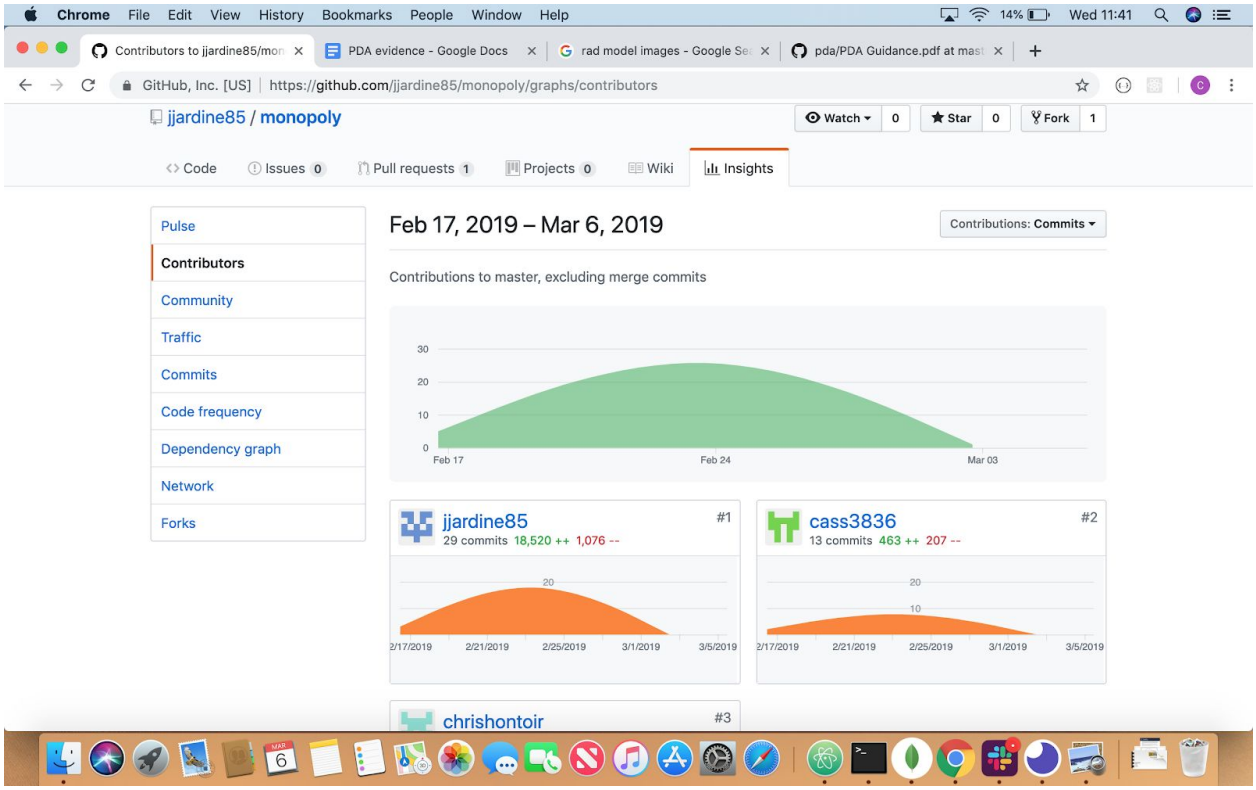
```
➜  PDA_Static_and_Dynamic_Task_A git:(master) ✗ ruby specs/testing_task_spec.rb
Run options: --seed 8767

# Running:

F.F..

Finished in 0.001263s, 3958.8282 runs/s, 3958.8282 assertions/s.

  1) Failure:
TestCardGame#test_check_for_ace_true [specs/testing_task_spec.rb:16]:
Expected: true
  Actual: false

  2) Failure:
TestCardGame#test_cards_total [specs/testing_task_spec.rb:32]:
Expected: "You have a total of 6"
  Actual: "You have a total of 15"

5 runs, 5 assertions, 2 failures, 0 errors, 0 skips
➜  PDA_Static_and_Dynamic_Task_A git:(master) ✗
```





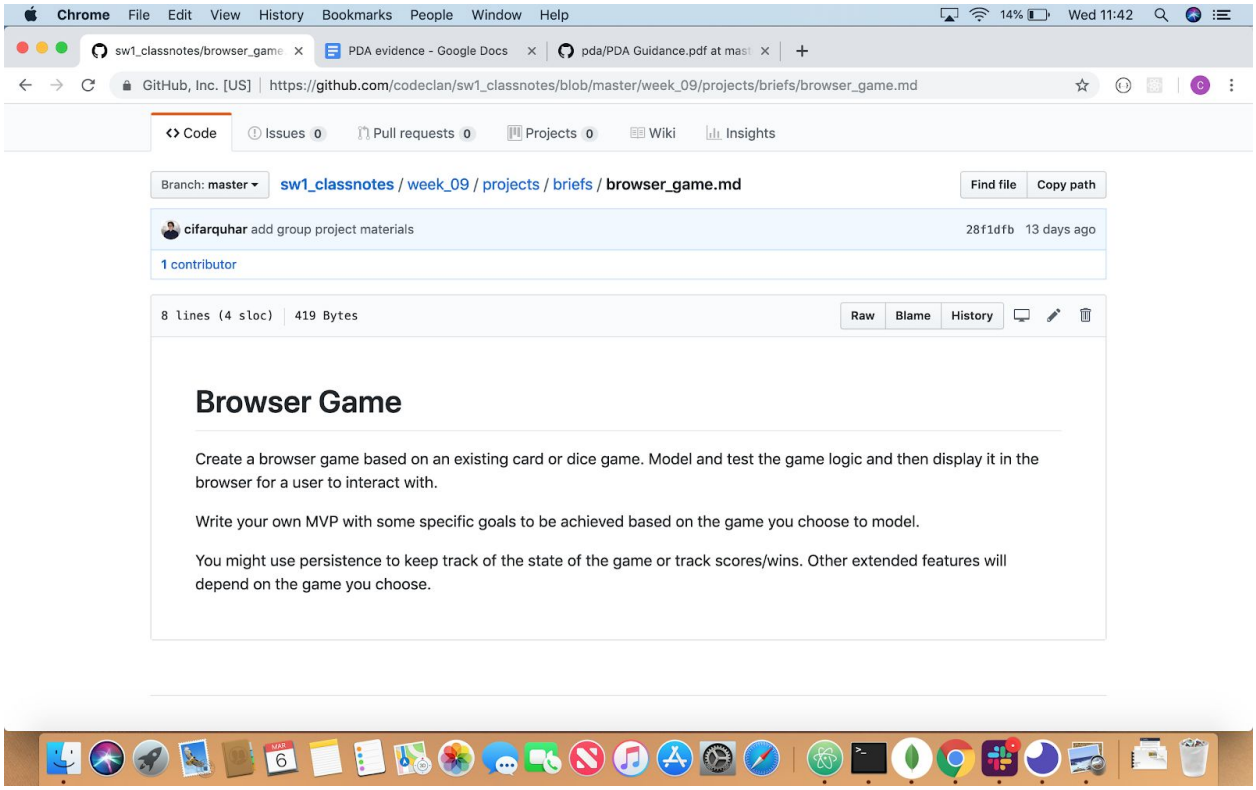
The above is four screenshots demonstrating the stages of TDD I completed as part of a card game. The first two screenshots show tests I wrote, and the terminal in which I ran the tests and two of them failed. The second two screenshots show the functions I was testing with the corrected errors and the terminal in which I ran the tests again and they passed, due to my corrections.

P.1. Github contributor's page:



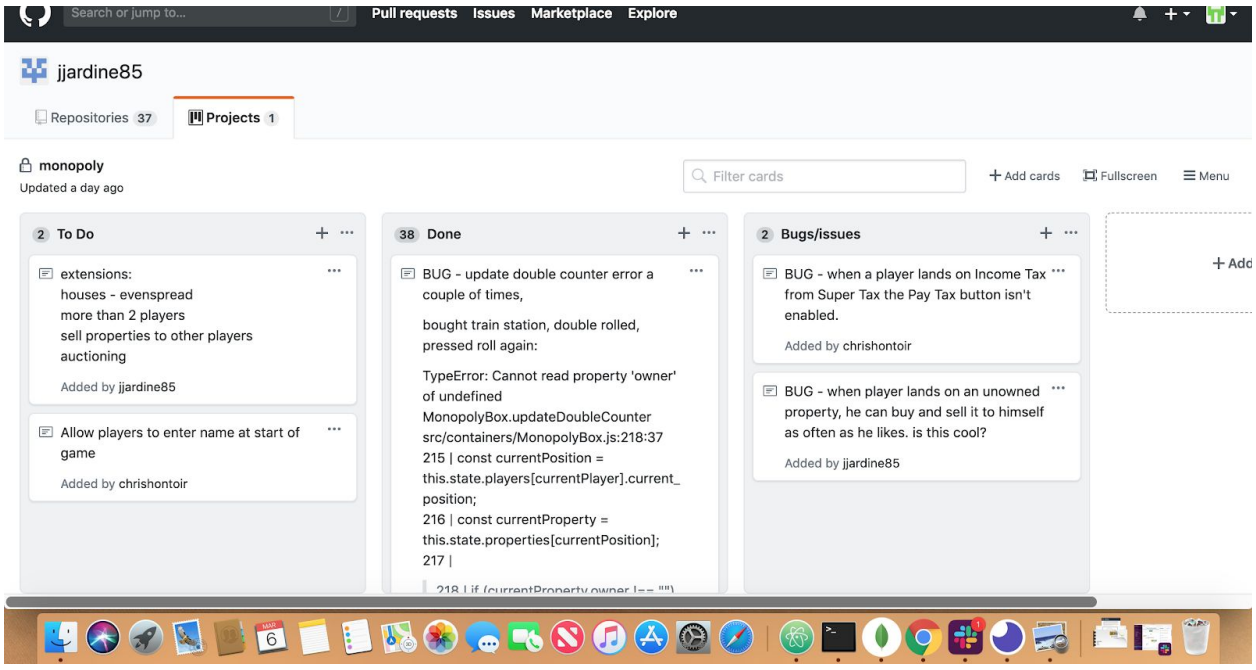
The above screenshot shows the contributor page for the Github repo from my group project. For the project, a group of 4 of us worked together to create an online version of the Monopoly board game. We used 3 of our accounts to push commits to the master branch, (but primarily we used jjardine85).

P.2. Group project brief:



Above is the official brief provided to my group by our instructors for our group project. We had a week to create a working version of a browser game that was based on an existing game. As mentioned above, we opted to re-create an online version of Monopoly. We created specific goals for our MVP, which can be seen in the photo below (under P.3).

P.3. Planning during group project:



The above screenshots demonstrate a few elements of my group's planning/development process.

- The first screenshot is a slide from the final presentation of our work. As can be seen, we had an initial plan for an MVP and extensions. During a retro meeting part-way through the week, we re-wrote our goals, as we had come on further than expected in the given

time, whilst also having reached certain barriers that we had not anticipated. As such, we wrote a new set of MVP/extension aims.

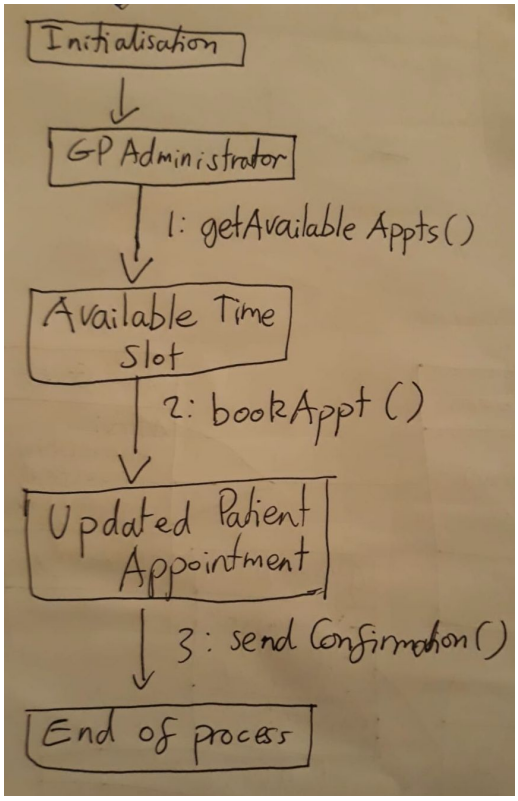
- The first screenshot also shows an initial, minimally-styled version of our board display, as well the final version.
- The second screenshot shows our Kanban board via which we tracked the completion state of different tasks, as well as various bugs that needed fixing.

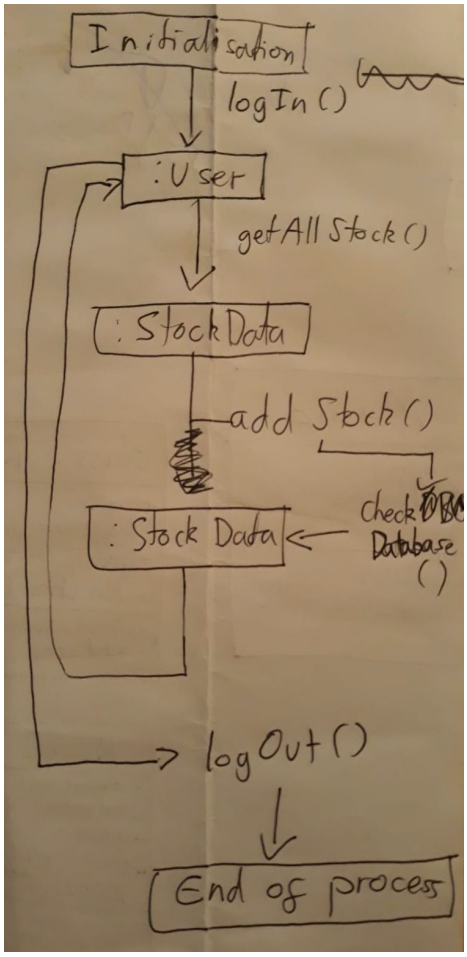
P.4. Acceptance criteria and test plan:

Acceptance Criteria	**Expected Result**	**Pass/Fail**
A user is able to view all stock in a single view.	The 'View all stock' link on the Homepage redirects to a table displaying all albums in alphabetical order, and their details.	Pass
A user is able to add stock to the database.	The 'Add album' link on the Homepage redirects to a list of all artists, which the user can click on to take them to the page of albums under that artist. Here they can input new album details in a form at the bottom of the page, submit the form, and see the new album displayed with the rest of the stock.	Pass
A user can only add an album to the database if the artist is already saved to the database.	The list of artists that the user is redirected to when they click 'Add album' has an option at the bottom to 'Add artist to database'. This button will redirect them to a table of all the artists, with a form underneath via which a new artist can be saved to the database. The user can then click on that artist and add an album in the normal way.	Pass
A user is able to edit stock once it has been added to the database.	On the filtered stock pages (i.e. on each artist's table of albums) each album has an 'Edit album' button, which redirects the user to a form that allows them to re-submit the details of that album. This overrides the original details.	Pass

Above is an acceptance criteria and test plan I wrote for my shop inventory project. It lays out various elements of functionality of my MVP, from the user's perspective (i.e. what a user should be able to do), and the manner in which the program allows the user to perform this functionality. All criteria were met so each row 'passes'.

P.7. System interaction diagrams:

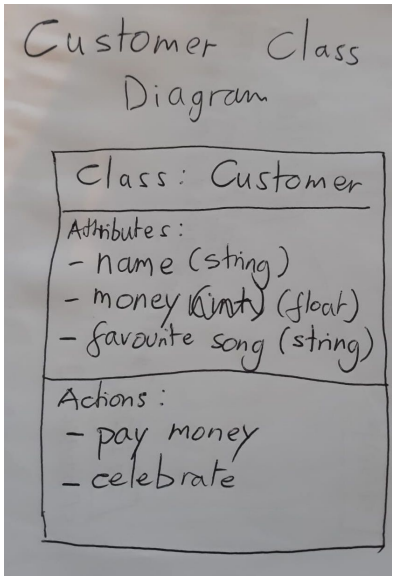


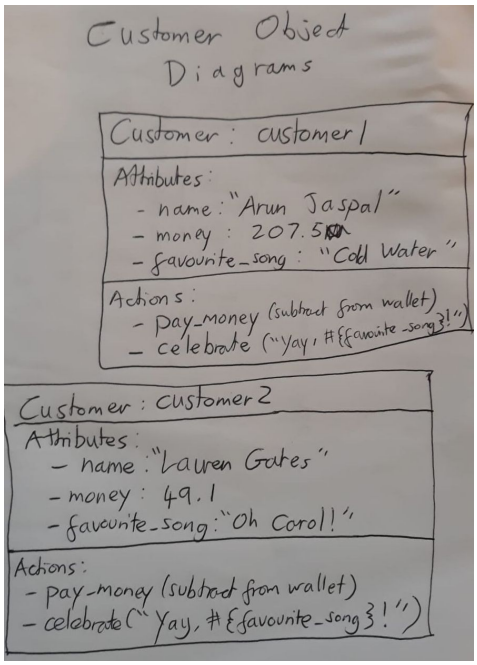


Above are two systems interaction diagrams I drew out to visualise the interaction between objects and the ‘messages’ passed between them.

- The first is a system in which a patient books a GP appointment. The system involves retrieving and editing/updating appointment data.
- The second is a system in which a shop manager views and adds stock to her shop inventory. The system involves a user retrieving and adding to stock data, as well as the program checking existing data to see if the added data already exists.
- Both diagrams display an initialisation as well as an end of the interaction process.

P.8. Object diagrams:





The photos above show a class diagram for a customer and, underneath, two object diagrams with instances of that customer class. The object diagrams provide actual values of the class attributes (here: name, money and favourite song), as well as the class methods and what they would return.

P.17. Bug tracking report:

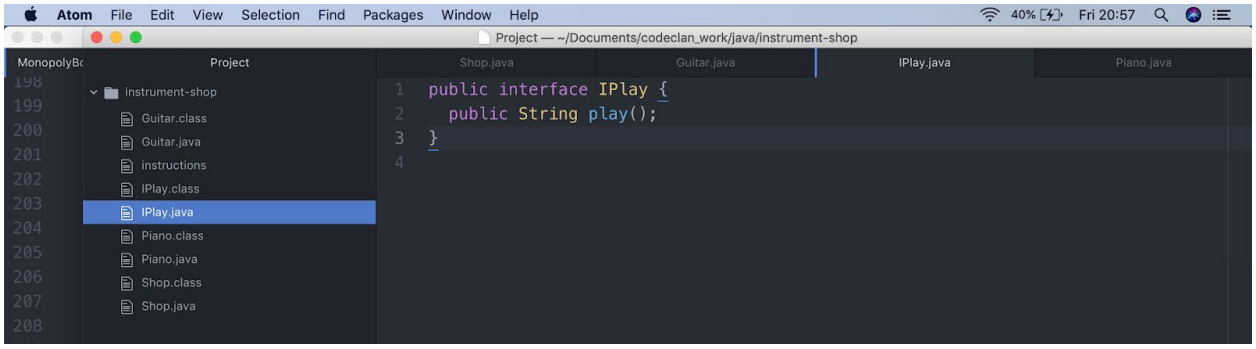
Bug/error	Solution	Date

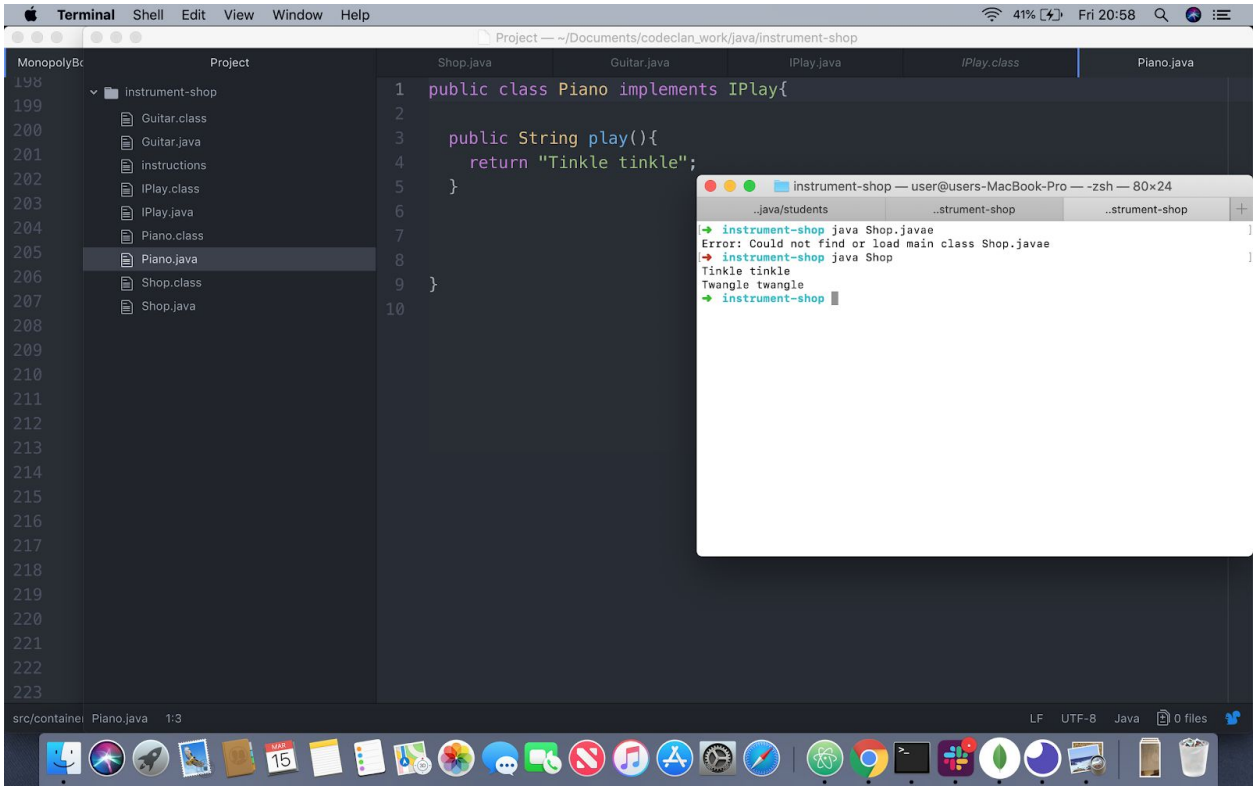
When a double is rolled when passing go the piece disappears.	Our playerMove function had the passGo function and the checkDoubleCounter in the wrong order, so it was trying to re-position the pieces in places that did not exist. We simply shifted the order of functions around.	26/02/2019
After mortgaging and unmortgaging several times, the players money turns into a float.	The calculation in the mortgaging functions did not round the numbers down before performing calculations. We added a .floor.	28/02/2019
If a player mortgages a property before the other player pays them rent, the pay rent button is disabled and the game gets stuck.	We disabled the mortgage button for current property until the pay rent button is pressed.	27/02/2019
A player is able to buy houses on utilities.	We added a conditional to adding of the buyHouses element that meant it was only added if the property set was neither stations nor utilities.	27/02/2019
When a player lands on an unowned property, they can buy and sell it ad infinitum.	We did not 'fix' this bug as it was not a priority because it did not actually do anything to break the game.	N/A

Above is a record I kept during my group project (creating a browser version of Monopoly) of some of the bugs/errors we encountered. The first column summarises the problem into which we ran. The second column details how we managed to resolve the problem. The third column shows the date it was solved (if it was solved). It was helpful to keep track of issues so that we could come back to solve them at a later point. It was also helpful to record how we solved the problems, so that we could learn better from our mistakes and fix bugs more quickly in the future, as we had an easy record of fixes to refer back to (and often bugs were quite similar).

I.T.7. Use of polymorphism:

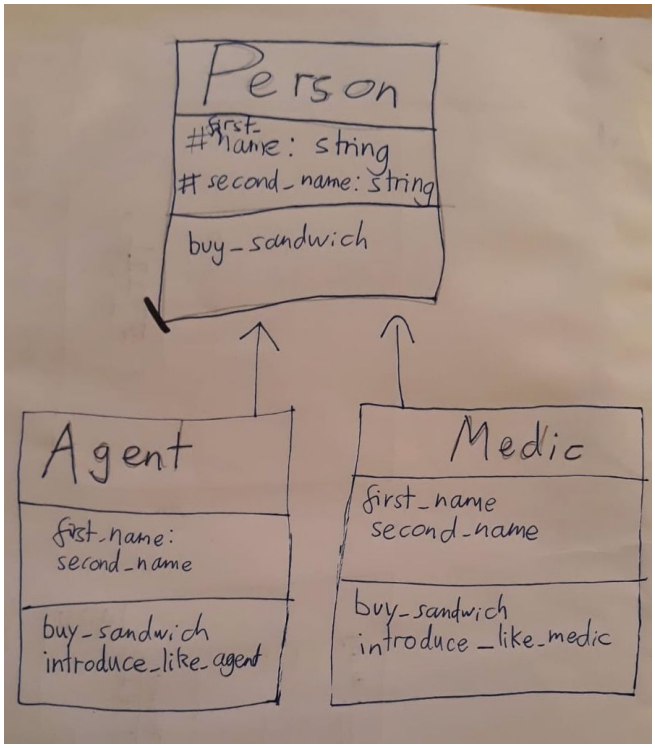

Project
instrument-shop
Guitar.class
Guitar.java
instructions
IPlay.class
IPlay.java
Piano.class
Piano.java
Shop.class
Shop.java
Shop.java
Guitar.java
IPlay.java
Piano.java
public class Shop {
private ArrayList<IPlay> stock;
private String shopName;
public Shop(String name){
this.shopName = name;
this.stock = new ArrayList<IPlay>();
}
public void addStock(IPlay stock) {
this.stock.add(stock);
}
public ArrayList<IPlay> getStock(){
return this.stock;
}
public static void main(String[] args){
Shop cassiaMusic = new Shop("CassiaMusic");
Piano p = new Piano();
Guitar g = new Guitar();
cassiaMusic.addStock(p);
cassiaMusic.addStock(g);
for (int i=0; i<cassiaMusic.getStock().size(); i++) {
System.out.println(cassiaMusic.getStock().get(i).play());
}
Shop.java* 22:48
LF UTF-8 Java 0 files


Atom File Edit View Selection Find Packages Window Help
40% Fri 20:57
Project — ~/Documents/codeclan_work/java/instrument-shop
Project
instrument-shop
Guitar.class
Guitar.java
instructions
IPlay.class
IPlay.java
Piano.class
Piano.java
Shop.class
Shop.java
Shop.java
Guitar.java
IPlay.java
Piano.java
public interface IPlay {
public String play();
}



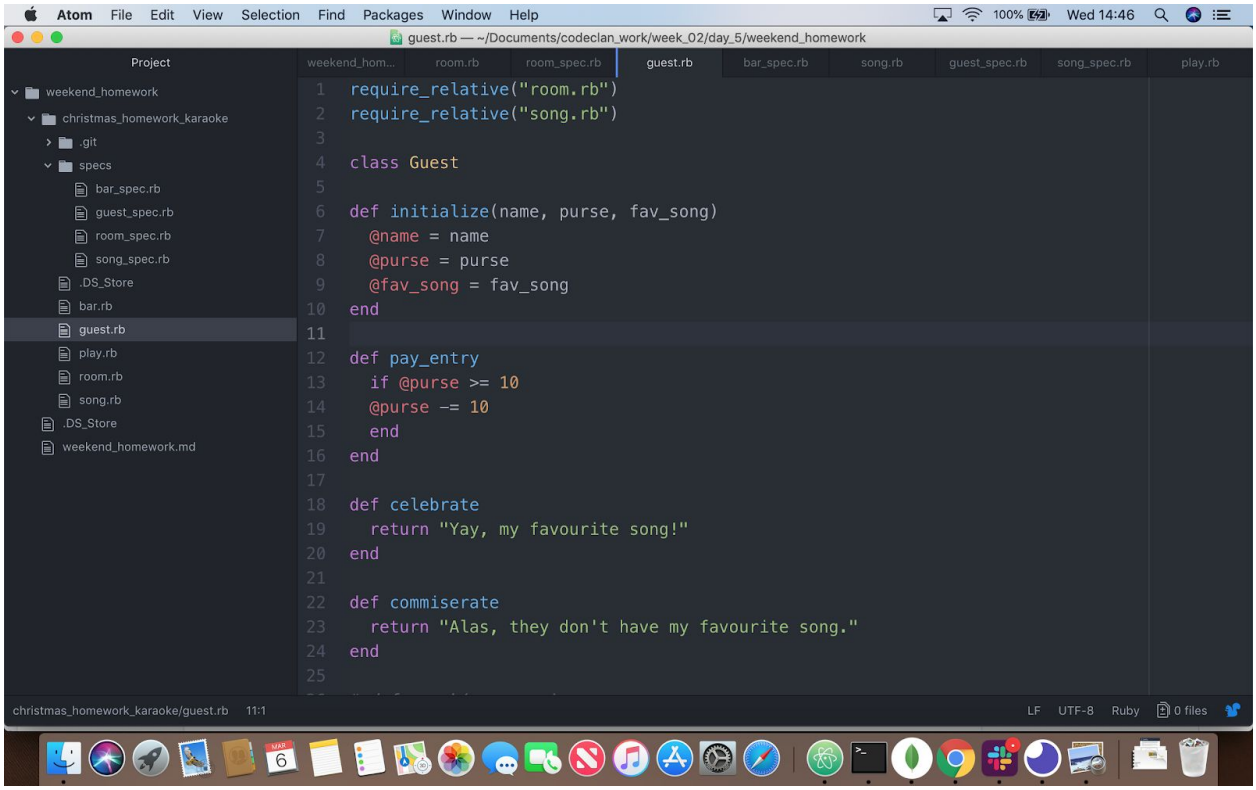
The above screenshots demonstrate the use of polymorphism in a Java program. As can be seen in the first screenshot, the shop included instruments that were instances of their instrument class (e.g. Piano or Guitar). Both, however, also implemented the same IPlay interface, despite being separate classes. The second two screenshots show the IPlay interface with its play method, and the Piano class with what this play method would return for that particular class. The third screenshot also shows the result in the terminal of the document in the first screenshot being run. It shows the output of both instruments calling their own IPlay method.

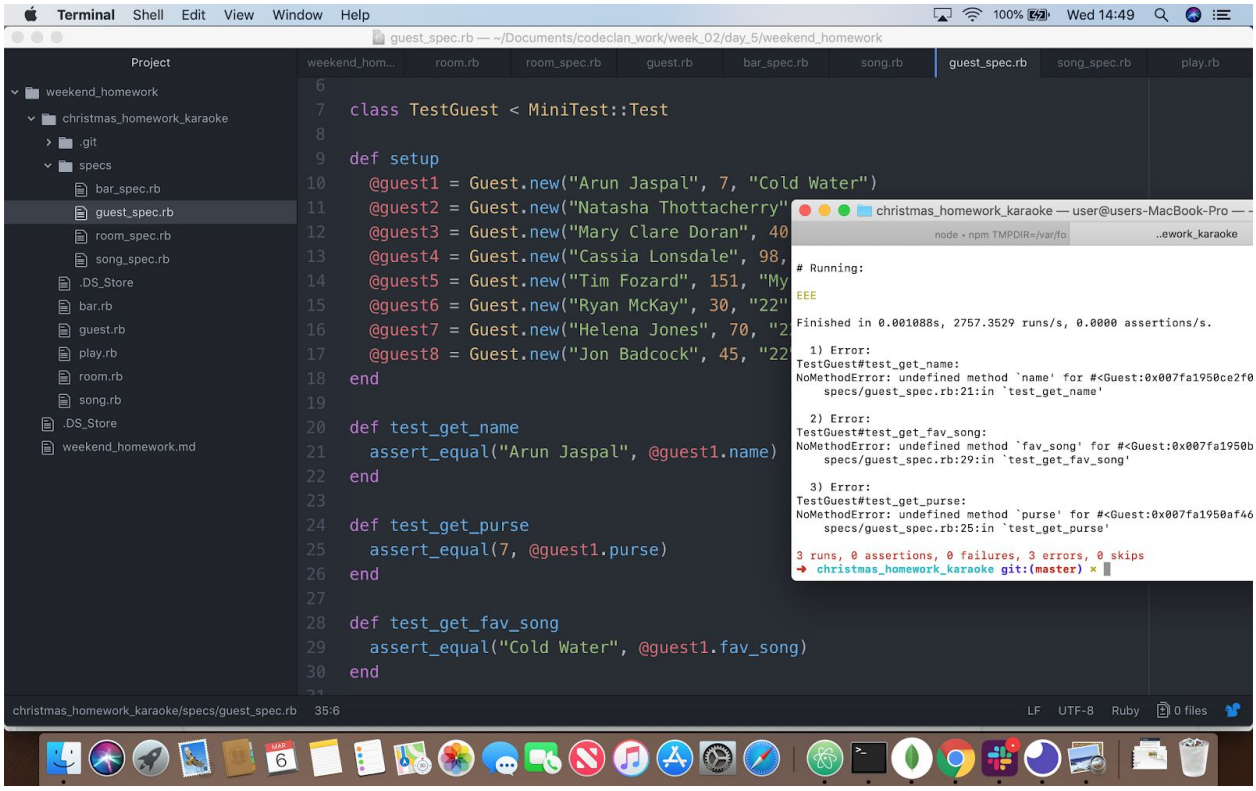
A.D.5. Inheritance diagram:

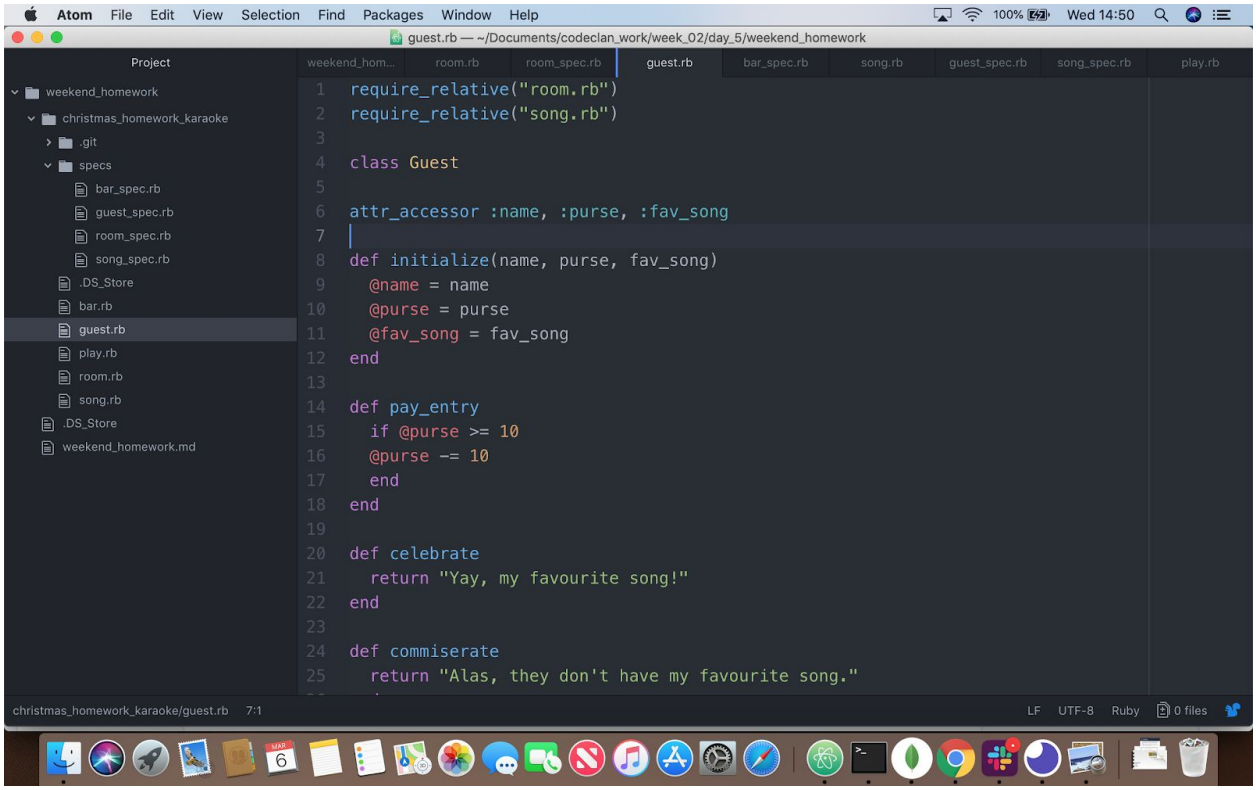


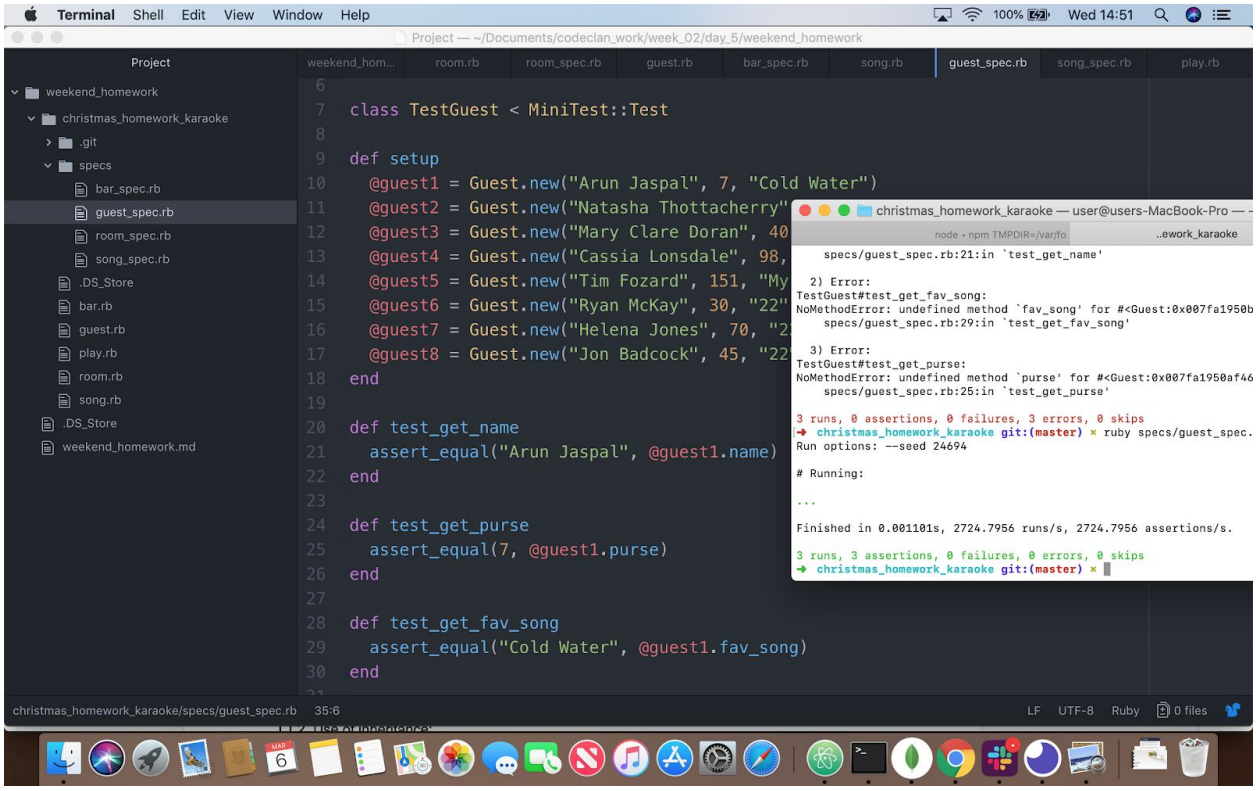
The above photo is an inheritance diagram I drew to help me visualise how the Agent class and Medic class would both inherit certain attributes and functions from the Person class (whilst also having their own specific divergences). The agent and medic inherit their first and last name attributes from the person class, but have separate introduction methods (that will use their names in different ways).

I.T.1 Use of encapsulation:


Atom File Edit View Selection Find Packages Window Help
Wed 14:46
guest.rb — ~/Documents/codeclan_work/week_02/day_5/weekend_homework
Project
weekend_homework
christmas_homework_karaoke
.git
specs
bar_spec.rb
guest_spec.rb
room_spec.rb
song_spec.rb
.DS_Store
bar.rb
guest.rb
play.rb
room.rb
song.rb
.DS_Store
weekend_homework.md
weekend_hom...
room.rb
room_spec.rb
guest.rb
bar_spec.rb
song.rb
guest_spec.rb
song_spec.rb
play.rb
require_relative("room.rb")
require_relative("song.rb")
class Guest
def initialize(name, purse, fav_song)
@name = name
@purse = purse
@fav_song = fav_song
end
def pay_entry
if @purse >= 10
@purse -= 10
end
end
def celebrate
return "Yay, my favourite song!"
end
def commiserate
return "Alas, they don't have my favourite song."
end
christmas_homework_karaoke/guest.rb 11:1
LF UTF-8 Ruby 0 files

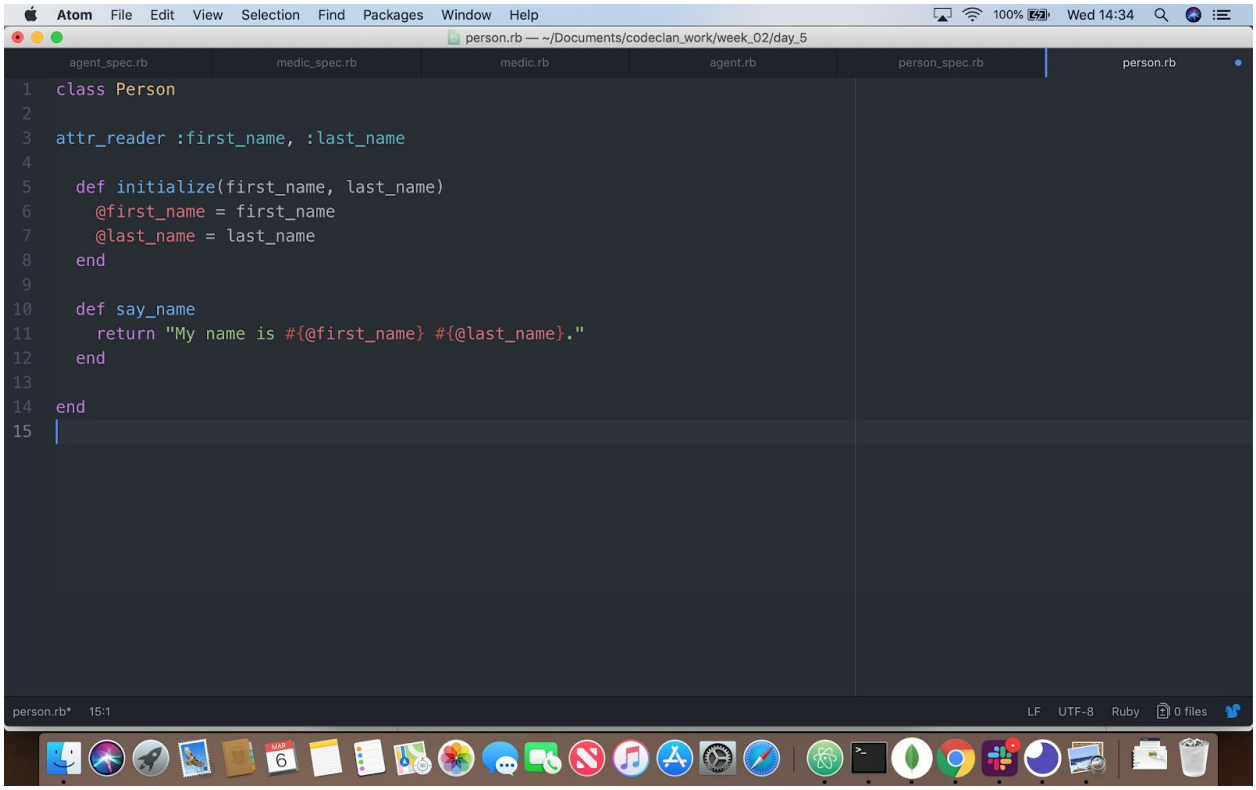

Terminal Shell Edit View Window Help
Wed 14:49
guest_spec.rb — ~/Documents/codeclan_work/week_02/day_5/weekend_homework
Project
weekend_homework
christmas_homework_karaoke
.git
specs
bar_spec.rb
guest_spec.rb
room_spec.rb
song_spec.rb
.DS_Store
bar.rb
guest.rb
play.rb
room.rb
song.rb
.DS_Store
weekend_homework.md
class TestGuest < MiniTest::Test
def setup
@guest1 = Guest.new("Arun Jaspal", 7, "Cold Water")
@guest2 = Guest.new("Natasha Thottacherry"
@guest3 = Guest.new("Mary Clare Doran", 40
@guest4 = Guest.new("Cassia Lonsdale", 98,
@guest5 = Guest.new("Tim Fozard", 151, "My
@guest6 = Guest.new("Ryan McKay", 30, "22"
@guest7 = Guest.new("Helena Jones", 70, "2
@guest8 = Guest.new("Jon Badcock", 45, "22
end
def test_get_name
assert_equal("Arun Jaspal", @guest1.name)
end
def test_get_purse
assert_equal(7, @guest1.purse)
end
def test_get_fav_song
assert_equal("Cold Water", @guest1.fav_song)
end
christmas_homework_karaoke — user@users-MacBook-Pro
Running:
EEE
Finished in 0.001088s, 2757.3529 runs/s, 0.0000 assertions/s.
1) Error:
TestGuest#test_get_name:
NoMethodError: undefined method `name' for #<Guest:0x007fa1950ce2f0
specs/guest_spec.rb:21:in `test_get_name'
2) Error:
TestGuest#test_get_fav_song:
NoMethodError: undefined method `fav_song' for #<Guest:0x007fa1950b
specs/guest_spec.rb:29:in `test_get_fav_song'
3) Error:
TestGuest#test_get_purse:
NoMethodError: undefined method `purse' for #<Guest:0x007fa1950af46
specs/guest_spec.rb:25:in `test_get_purse'
3 runs, 0 assertions, 0 failures, 3 errors, 0 skips
christmas_homework_karaoke git:(master) ×
christmas_homework_karaoke/specs/guest_spec.rb 35:6
LF UTF-8 Ruby 0 files

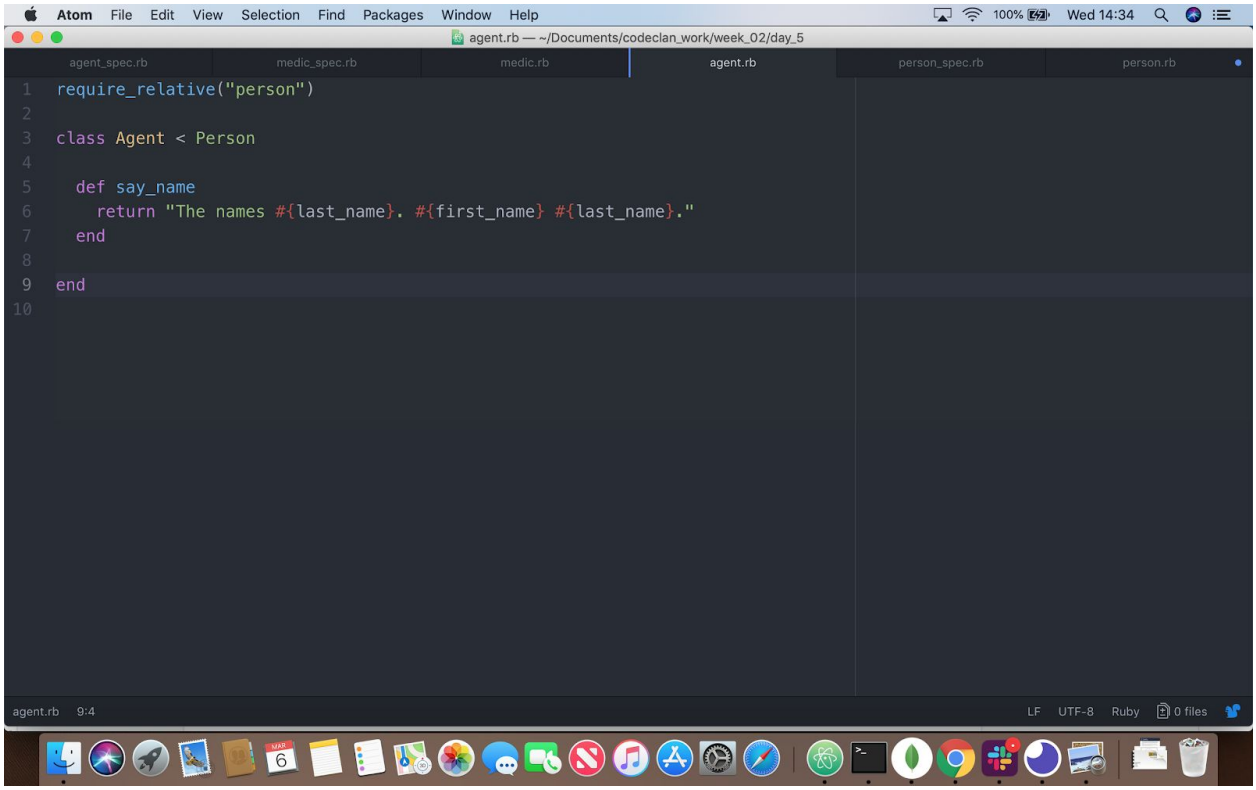

guest.rb — ~/Documents/codeclan_work/week_02/day_5/weekend_homework
Project
weekend_homework
christmas_homework_karaoke
.git
specs
bar_spec.rb
guest_spec.rb
room_spec.rb
song_spec.rb
.DS_Store
bar.rb
guest.rb
play.rb
room.rb
song.rb
.DS_Store
weekend_homework.md
require_relative("room.rb")
require_relative("song.rb")
class Guest
attr_accessor :name, :purse, :fav_song
def initialize(name, purse, fav_song)
@name = name
@purse = purse
@fav_song = fav_song
end
def pay_entry
if @purse >= 10
@purse -= 10
end
end
def celebrate
return "Yay, my favourite song!"
end
def commiserate
return "Alas, they don't have my favourite song."
christmas_homework_karaoke/guest.rb 7:1
LF UTF-8 Ruby 0 files

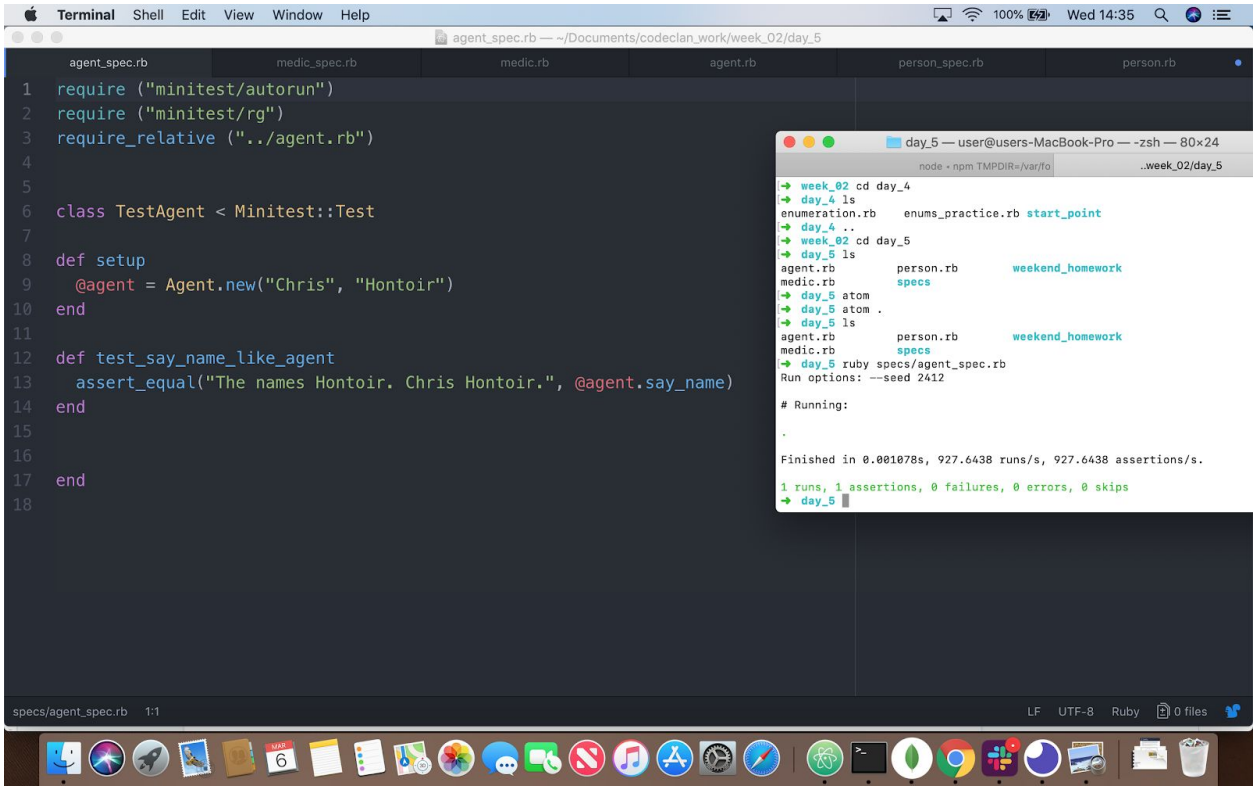

Project — ~/Documents/codeclan_work/week_02/day_5/weekend_homework
guest_spec.rb
class TestGuest < MiniTest::Test
def setup
@guest1 = Guest.new("Arun Jaspal", 7, "Cold Water")
@guest2 = Guest.new("Natasha Thottacherry"
@guest3 = Guest.new("Mary Clare Doran", 40
@guest4 = Guest.new("Cassia Lonsdale", 98,
@guest5 = Guest.new("Tim Fozard", 151, "My
@guest6 = Guest.new("Ryan McKay", 30, "22"
@guest7 = Guest.new("Helena Jones", 70, "2
@guest8 = Guest.new("Jon Badcock", 45, "22
end
def test_get_name
assert_equal("Arun Jaspal", @guest1.name)
end
def test_get_purse
assert_equal(7, @guest1.purse)
end
def test_get_fav_song
assert_equal("Cold Water", @guest1.fav_song)
end
christmas_homework_karaoke — user@users-MacBook-Pro
specs/guest_spec.rb:21:in `test_get_name'
2) Error:
TestGuest#test_get_fav_song:
NoMethodError: undefined method `fav_song' for #<Guest:0x007fa1950b
specs/guest_spec.rb:29:in `test_get_fav_song'
3) Error:
TestGuest#test_get_purse:
NoMethodError: undefined method `purse' for #<Guest:0x007fa1950af46
specs/guest_spec.rb:25:in `test_get_purse'
3 runs, 0 assertions, 0 failures, 3 errors, 0 skips
christmas_homework_karaoke git:(master) ruby specs/guest_spec.
Run options: --seed 24694
Running:
Finished in 0.001101s, 2724.7956 runs/s, 2724.7956 assertions/s.
3 runs, 3 assertions, 0 failures, 0 errors, 0 skips
christmas_homework_karaoke git:(master)
christmas_homework_karaoke/specs/guest_spec.rb 35:6
LF UTF-8 Ruby 0 files

- The above screenshots demonstrate encapsulation. The first two screenshots show the user attempting to access the attributes of the Guest class (screenshot 1), from a different document and failing (screenshot 2, console).
- The second two screenshots show the same code but with attribute reader methods added (screenshot 3, line 6), which, if called, allow the user to read (or re-write) those attributes. The same tests now pass (screenshot 4, console).

I.T.2. Use of inheritance:



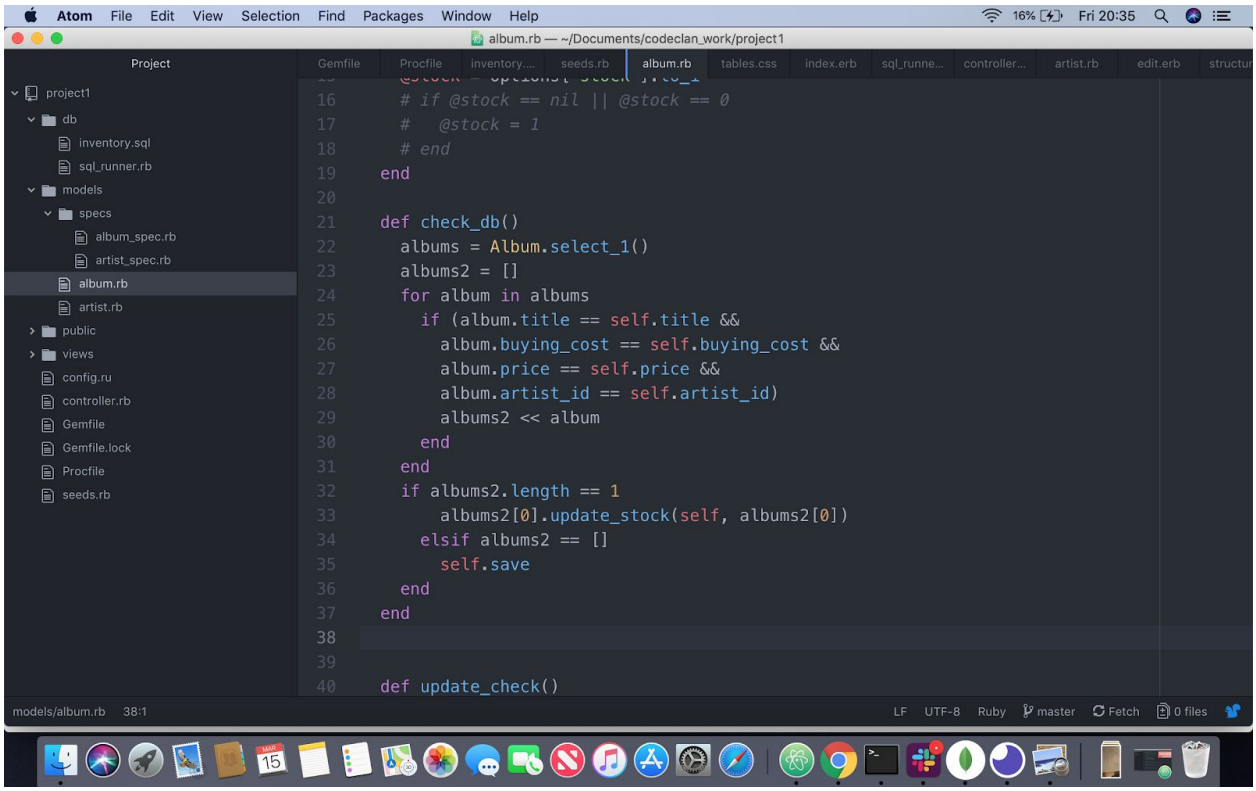


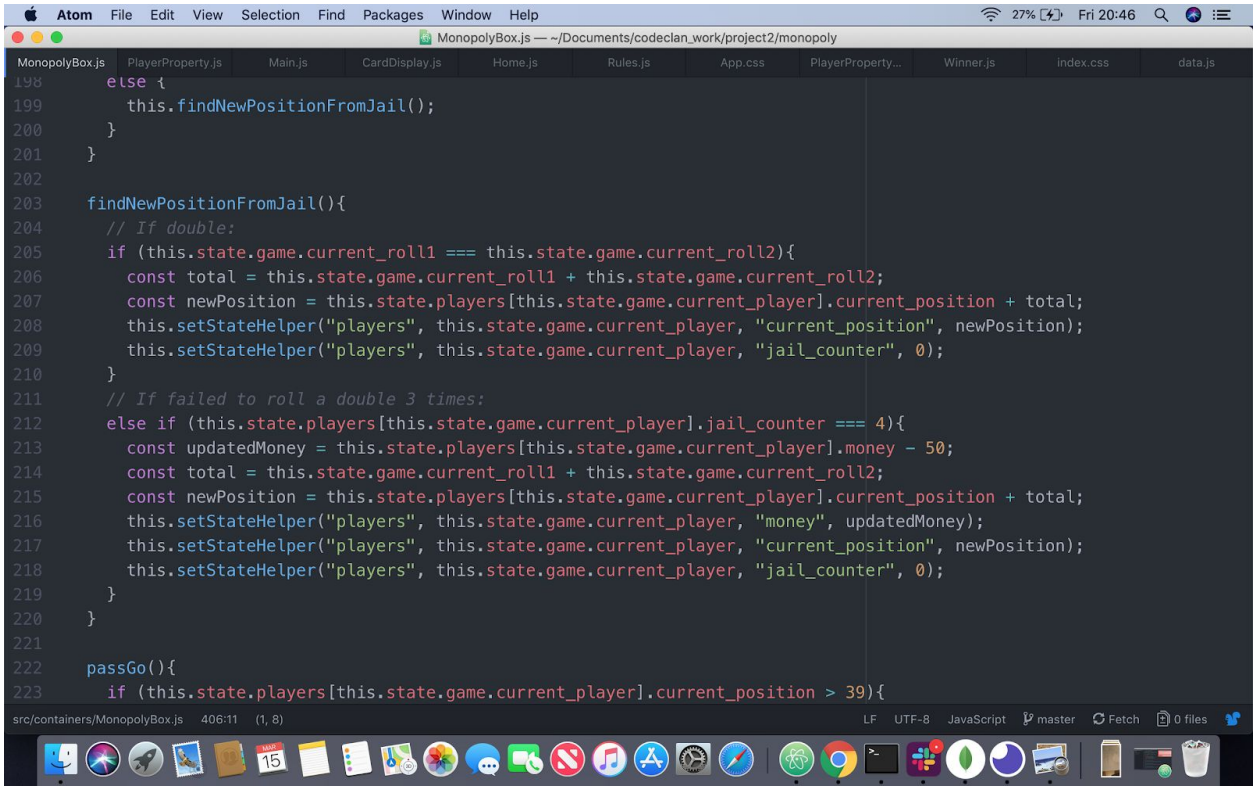


The above screenshots demonstrate the use of class inheritance in a program.

- Screenshot 1 shows the Person class, which has the attributes of first_name and last_name (lines 5-8).
- Screenshot 2 shows the Agent class, which inherits from the Person class (line 3). Agent also has its own say_name method (line 6), which is different from the Person say_name method. (The Agent's say_name function still uses first_name and last_name, even though these attributes have not been explicitly initialized in the Agent class.)
- Screenshot 3 demonstrates the function working (see passing test in the console) by creating a new Agent that takes in the first_name and last_name parameters defined in the Person class. (The Agent defines no parameters explicitly, but takes in these names because it inherits from the Person class.)

P.9. Algorithms:





Above are screenshots of two algorithms I wrote.

The first is an algorithm I wrote for my shop inventory project. The purpose of the function is to take user input and (ultimately) save it to a database:

- The function is called on the user input.
- It first retrieves all existing albums from the database (line 22).
- It then creates an empty array (line 23).
- It them loops through all the retrieved albums and, if the details match the details in the user input, it pushes that album onto the empty array (lines 24-31).
- If something does get pushed to the array by the end of the loop, it means the album already exists in the database, so it is not saved again, but instead has its stock-count updated (line 33 - this functionality is written in a separate ‘update_stock’ method).
- If the array is empty by the end of the loop, then the album does not already exist, and so the user input is saved to the database (line 35).

The second is an algorithm I wrote for my group’s Monopoly game project. The purpose of the function is to find a players new position on the board in the case of them being in jail:

- The function is called if the current player is in jail.
- The function checks to see if the current players dice roll is a double (line 205). If it is, the player can move out of jail, so it updates the state of the player and changes their position to their new position (jail plus the dice roll) (line 206-208). It also resets their ‘jail counter’ to 0 (i.e. how many goes they have been in jail) (line 209).

- If the player does not roll a double but they have been in jail 3 goes in a row (checked in line 212), they must pay to get out. The function therefore updates the state of the player by taking 50 away from their money (lines 213 and 216) and moving them out of jail to their new position based on their dice roll (lines 214-215 and 217). Again, their jail counter is reset 0 (line 218).
- If neither of the above conditions are met, nothing happens.