Donation Blog

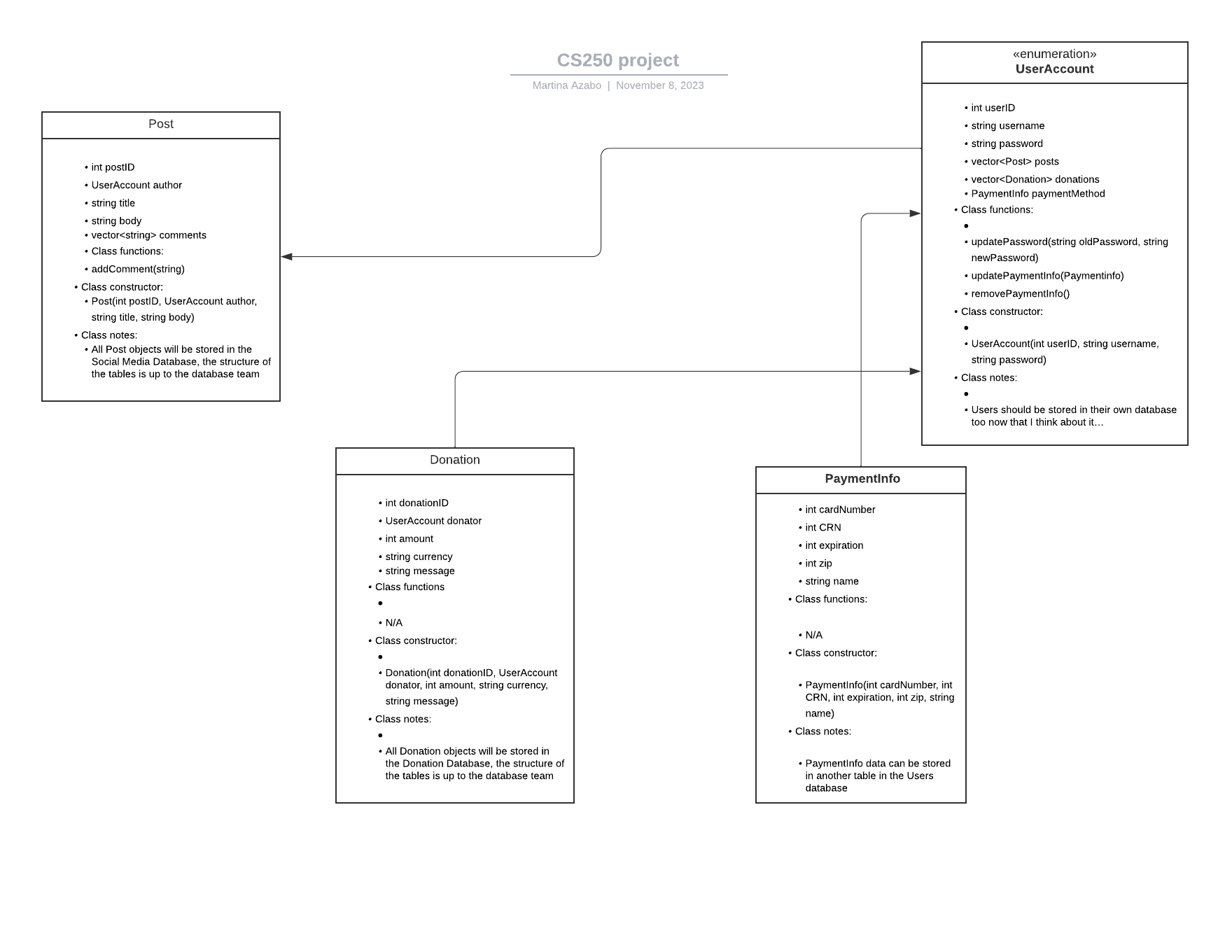
Software Design Specifications

Martina Azabo, Htet Hnin Su Wai, Nicholas Dibello-Hitta, Juan Diaz

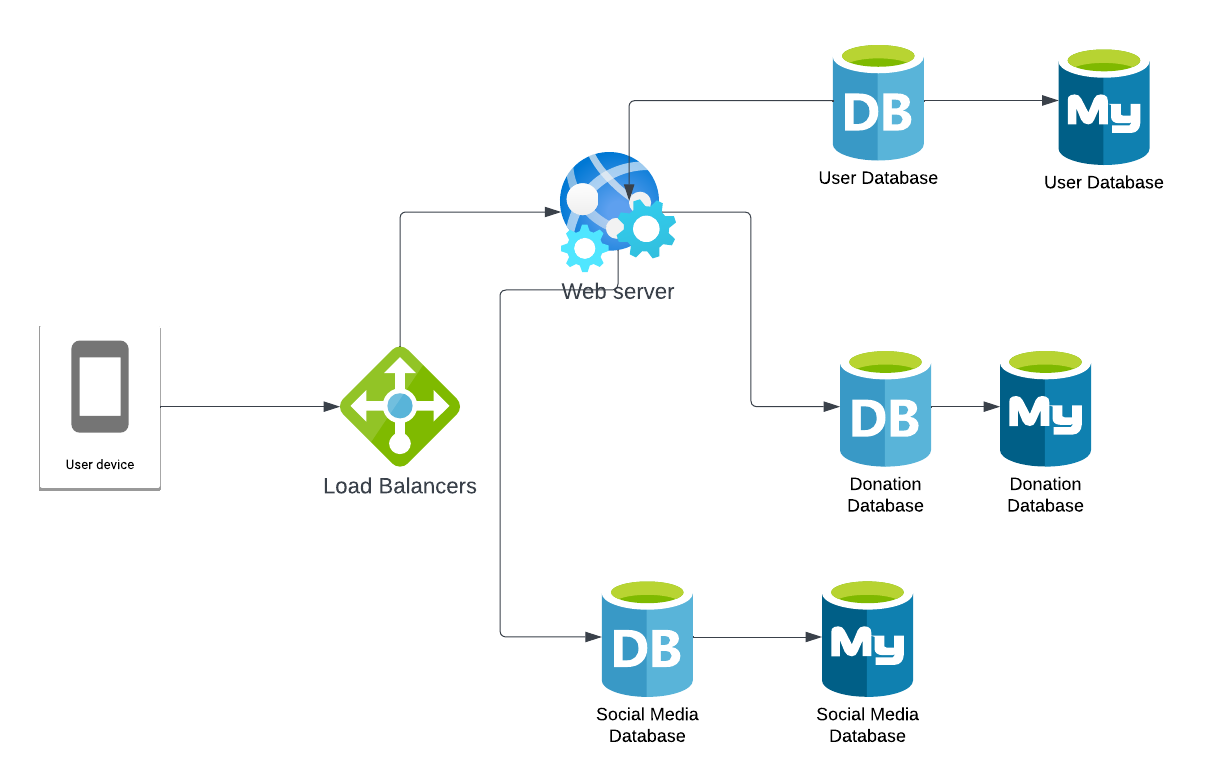
**System Description**

* The system we’re developing is a donation website. The main feature of this website is allowing users to browse through donation prompts posted by admin users and navigating to a payment tab to donate amount they desire. Users will land on a log in page on the landing page where they’ll be able to create an account where they’re able to keep tab of their donations and search for topics or locations that are relevant to them. Users will also be able to share these donation prompts via sharing a link or QR Code or by sharing it on social media platforms. Users will also be able to log out, delete accounts and customize their search results to find the donation best suitable for them. Admin users will be able to log in and out, post prompts, keep track of donations received, delete, or edit prompts and post updates regarding the donation project and how it’s progressing.

**Software Architecture Overview**



**Software Architecture Descriptions**



*Class Diagram*

sharePost()

* Bool class that displays icons to automatically share donation page to other websites
* Bool sharePost: reveals a hidden portion of the page to allow quick sharing on other platforms or a shortened url
* Bool shareQRCode: upon the user making this true, a QR Code is generated for that specific page.

donationType()

* String class containing information about the each fundraiser that is used to filter out different kinds of fundraisers
* Int zipCode: entered to indicate what area the fundraiser originates from
  + Converts to string when used to filter
  + Used with User Account to find fundraisers at or nearby user’s entered zip code.
* String category: Describes what the fundraiser is for in a broader form
  + i.e. Medical, Political, Religious, Personal, Construction.

paymentInfo()

* String paymentMethod: Indicates what type of card is being used
  + i.e. Visa, MasterCard, American Express
* int transactionID: gives each donation a specific ID for referencing and proof
* bool PurchaseStatus: if payment is authorized and complete, this will become true, if not it remains false.
  + Used to indicate if a donation is incomplete
* Int confirmDonation: used to show the user how much they are about to donate, and to confirm if they wish to donate that specific amount.

UserAccount:

* String Username: 1st part of user’s login credentials set at creating an account. The user’s email address can also satisfy being the username.
* String password: 2nd part of user’s login credentials set when creating an account.
  + Is encrypted upon creation
* Obj reaction: Is called when the user wishes to “like” or indicate acknowledgement with an emoji to express their thoughts.
  + i.e. Facebook reactions to a post
* int zipCode: user’s entered zip code relative to their home.
  + Used to search for fundraisers near their area
* Bool donationHistory: if user wishes, they can publicly display which fundraisers they’ve contributed to
* Void addComment(comment): A method to take a user generated comment and post it onto the page they are on
  + Comment is a string
* Void addReaction(reaction): A method that posts what the user pushed on obj reaction and post it to the analytics of the fundraiser.

AdminUser:

* Void addPrompt(): A method allowing those users of Admin level to create the fundraiser.
* Int donationsRecieve: shows how much money has been contributed to the fundraiser. Is also reflected on admin user’s page to see how much funds were accumulated.
* Void deletePrompt(): A method that allows admin users to delete fundraisers created by them.
* Void editPrompt(): A method that allows admin users to edit information from the fundraiser page.
  + Used in conjunction with void postUpdates()
* Void postUpdates(): A method that takes the changes made by the admin user and uploads the changes to the page.

*Architecture Diagram*

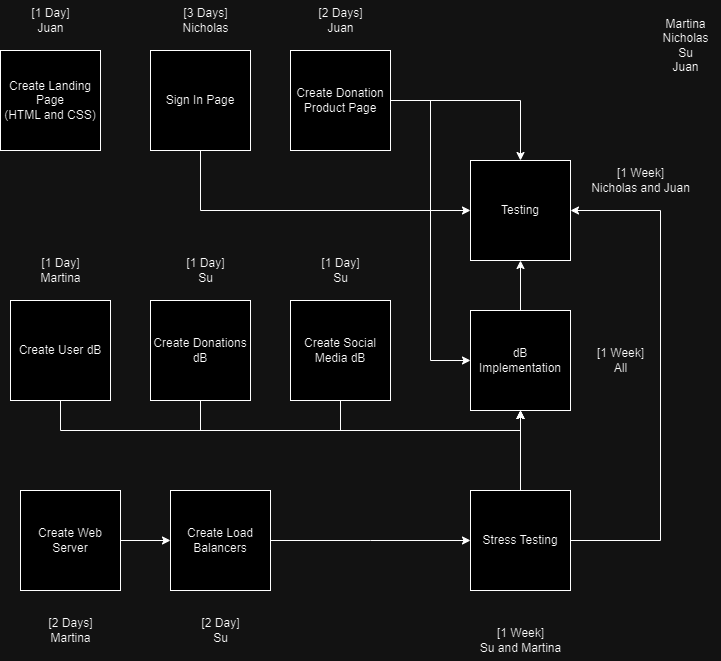
* User Device: any device that allows either consumers or administrators to access the website
* Load Balancers: protects servers from being overloaded and DDoS attacks.
* Web Server: main web hosting server for the page.
* Social Media Database: hosts information on the sites’ API data and usage of other sites
* Donation Database: contains information on fundraisers currently present on site. Each entry is removed if the admin user who made it requests it to be deleted.

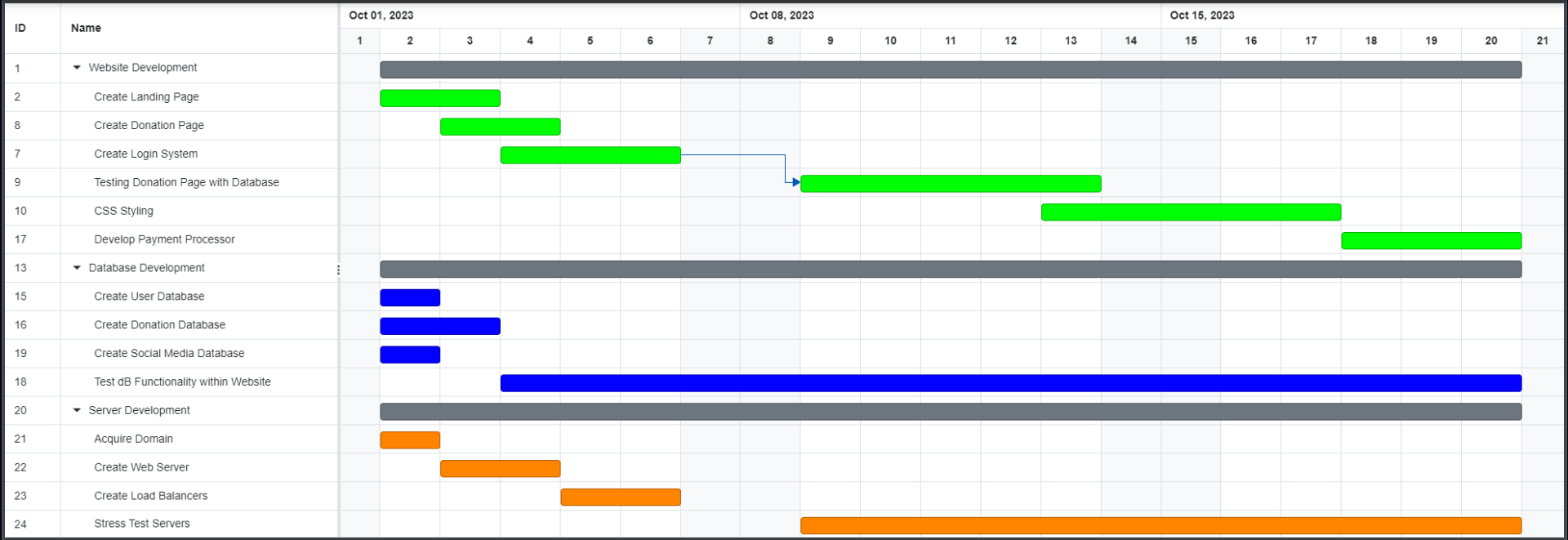
*Attributes*

* Responsive website that can be easily viewed on any device
* Built so that the system can be converted into a mobile phone app at a later date
* Fast response time when accessing different pages and updates to the database
* Uses HTML5, CSS, and mySQL
* Two types of users
  + Consumers: those who contribute to fundraisers and view publicly available information
  + Administrators: higher access users who create fundraisers and manage them using the tools provided by the website
    - Must be approved by managers of the website.
* Payment process is done by a trusted third party.
* User information, including but not limited to:
  + Passwords
  + Payment Information

will be encrypted to increase security

**Development Plan and timeline**

****

****

Unit tests for functions:

UserAccount:

* addComment()
  + <addComment(“testComment”), comment == “testComment”>
  + <addComment(NULL), comment == ERROR>
* addReaction()
  + <addReaction(“:smile:”), reaction == “:smile:”>
  + <addReaction(“:frown:”), reaction == “:frown:”>

AdminUser:

* add prompt()
  + <add prompt(“testPrompt”), prompt == “testPrompt”>
  + <add prompt(NULL), prompt == ERROR>
* edit prompt()
  + <edit prompt(“modifiedPrompt”), prompt == “modifiedPrompt”>
  + <edit prompt(NULL), prompt == ERROR>
* delete prompt()
  + <delete prompt(“modifiedPrompt”), prompt == NULL>
  + <delete prompt(“testPrompt”), prompt == ERROR>
* postUpdates()
  + <postUpdates(prompt), updates == [prompt]>
  + <postUpdates(NULL), updates == ERROR>

Integration tests for classes:

userAccount test 1:

userAccount U

String comment = “testComment”

U.comment = “”

If (U.comment == “”)

U.addComment(comment)

else

U.addComment(“”)

temp = U.comment

return (comment == temp)

userAccount test 2:

userAccount U

String reaction = “testReaction”

U.reaction = “”

If (U.reaction == “”)

U.addReaction(reaction)

else

U.addReaction(“”)

temp = U.reaction

return (reaction == temp)

AdminUser test 1:

AdminUser A

String promptText = “testPrompt”

A.prompt = “”

If (A.prompt == “”)

A.add prompt(promptText)

Else

A.add prompt(“”)

Temp = A.prompt

Return (promptText == Temp)

AdminUser test 2:

AdminUser A

String promptText = “modifiedPrompt”

A.prompt = “originalPromptText”

If (A.prompt == “originalPromptText”)

A.edit prompt(promptText)

Else

A.edit prompt(“wrong”)

Temp = A.prompt

Return (promptText == Temp)

AdminUser test 3:

AdminUser A

String promptText = “promptText”

A.prompt = promptText

If (A.prompt == promptText)

A.delete prompt(promptText)

Else

A.edit prompt(“wrong”)

Temp = A.prompt

Return (promptText == Temp)

PaymentInfo test

Integration - confirmDonation

paymentMethod string

confirmDonation [1] = paymentMethod.confirmDonation[1]

if (confirmDonation = 1)

PurchaseStatus= true

else

PurchaseStatus= false

return (confirmDonation[1].status == true)

**System Testing**

//user is a normal user who is about to complete a transaction

System.user1 = new System;

user1.userAccount{

String Username = “John “;

String password = “ Smith “;

String comment = NULL;

Int zipCode = 92020;

Bool donationHistory: False;

}

user1.paymentInfo{

String paymentMethod = “VISA”;

Int transactionID = 65341;

Bool purchaseStatus = NULL;

Int confirmDonation = 0;

}

user1.sharePost{

Bool sharePost = False;

Bool shareQRCode = false;

}

}

System.admin1 = new System;

admin1.AdminUser{

Int donationsRecieve = 340;

String Username = “johnDoe123”;

String password = “safe\_password”;

Void addPrompt{

String title = “Funding for my new business”;

String fullName = “John Doe”;

String description = “Lorem ipsum dolor sit”;

}

Void deletePrompt{

Bool delete = False;

}

Void editPrompt{

Bool confirmEdit = False;

If (confirmEdit == True){

userInput -> String updTitle;

userInput -> String updDesc;

}else{

//nothing happens

}

}

Void postUpdates{

Bool updated = editPrompt(confirmEdit);

if(updated == True){

addPrompt(title) = editPrompt(updTitle);

addPrompt(description) = editPrompt(updDesc);

}else{

//nothing happens

}

}

admin1.donationType{

Int zipCode = 92020;

String category = “Small Businesses”;

}

//admin1 previously set above

**Test Case 1**

//user1 begins transaction for a donation

//output information with success or errors

user1.paymentInfo(confirmDonation) = 100 -> success

user1.paymentInfo(paymentMethod) = NULL ->error

user1.paymentInfo(PurchaseStatus) = False -> Success;

user1.paymentInfo(transactionID) = 456804 //randomly generated -> success

Print (“ERROR”);

admin1.addPrompt(“Funding for my new business”, “John Doe”, “Lorem ipsum dolor sit”) -> success

admin1.editPrompt(False); -> success

admin1.deletePrompt(False) -> success

admin1.PostPrompt(False) -> success

Print(admin1.addPrompt(title), admin1.addPrompt(name), admin1.addPrompt(description))

//success

Overall: ERROR due to incomplete transaction

**Test Case 2**

//user1 begins transaction for a donation

//output information with success or errors

user1.paymentInfo(confirmDonation) = 400 -> success

user1.paymentInfo(paymentMethod) = MasterCard -> success

user1.paymentInfo(PurchaseStatus) = True -> success

user1.paymentInfo(transactionID) = 178664//randomly generated -> success

Print (“User1 donated” + user1.paymentInfo(confirmDonation) + “with a “ + user1.paymentInfo(paymentMethod)+ “card. Payment confirmed”); -> success

admin1.addPrompt(“Funding for my new business”, “John Doe”, “Lorem ipsum dolor sit”) -> success

admin1.editPrompt(False); -> success

admin1.deletePrompt(False) -> success

admin1.PostPrompt(False) -> success

Print(admin1.addPrompt(title), admin1.addPrompt(name), admin1.addPrompt(description))

//success

Overall: Successful transaction. Payment sent through

**Data Management Strategy**

We will be using SQL Data Management for this website. Since our data is structured and the relationships between each data such as User information, Payment information and sharing post are consistent, it is suitable for us to use SQL databases to manage our data. Information such as payment transactions are important to be stored securely in a database. Due to the simplicity of our website, I think creating one database with several tables that correspond to each class will help us manage it successfully.