1.) List the features that were implemented (table with ID and title).
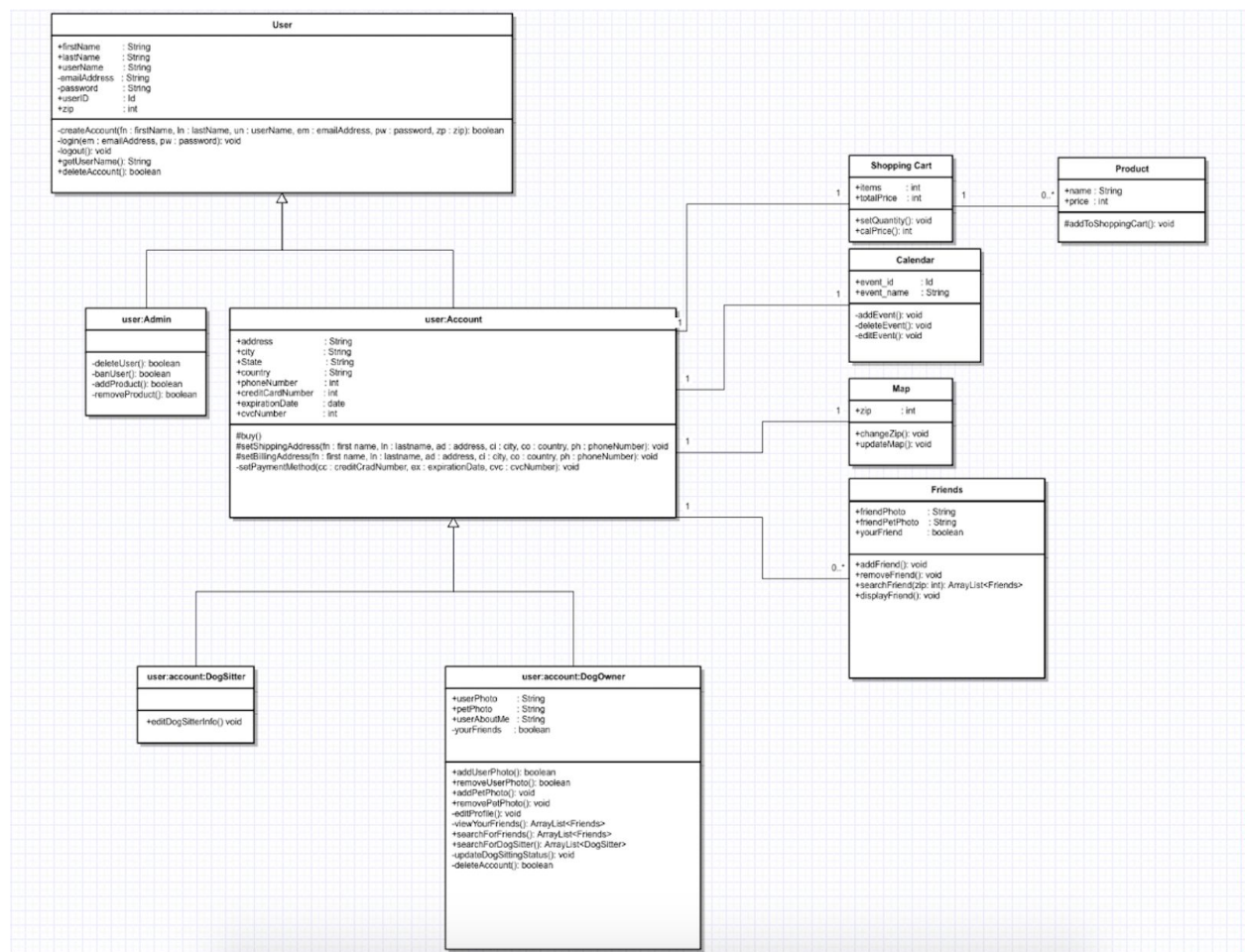
ID	Requirement	Topic area	Actor	Priority
US-01	As a user, I want to be able to sign up.	Authentication	User	High
US-02	As a user, I want to be able to login to my account.	Authentication	User	High

US-05	As a user, I want to be able to search for my friends.	Search	User	Medium
US-07	As a user, I want to be able to delete my friend from friend list.	Friends List	User	Low

US-08	As a user, I want to be able to search for pet sitters around me.	Search	User	Critical
US-09	As a user, I want to be able to buy pet tags.	Purchase	User	Medium
US-10	As a user, I want to be able to add pet tags to my shopping cart.	Purchase	User	Medium

US-12	As a user, I want to be able to add/remove events from calendar.	Event	User	Low
US-15	As a sitter, I want to be able to set up my online status.	Profile	Sitter	High
US-16	As a sitter, I want to be able to add contact information to profile.	Profile	Sitter	High

US-19	As a user, I want to be able to search for people to become friend.	Search	User	Medium
US-20	As a user, I want to be able to set up my dog sitting status.	Profile	User	High

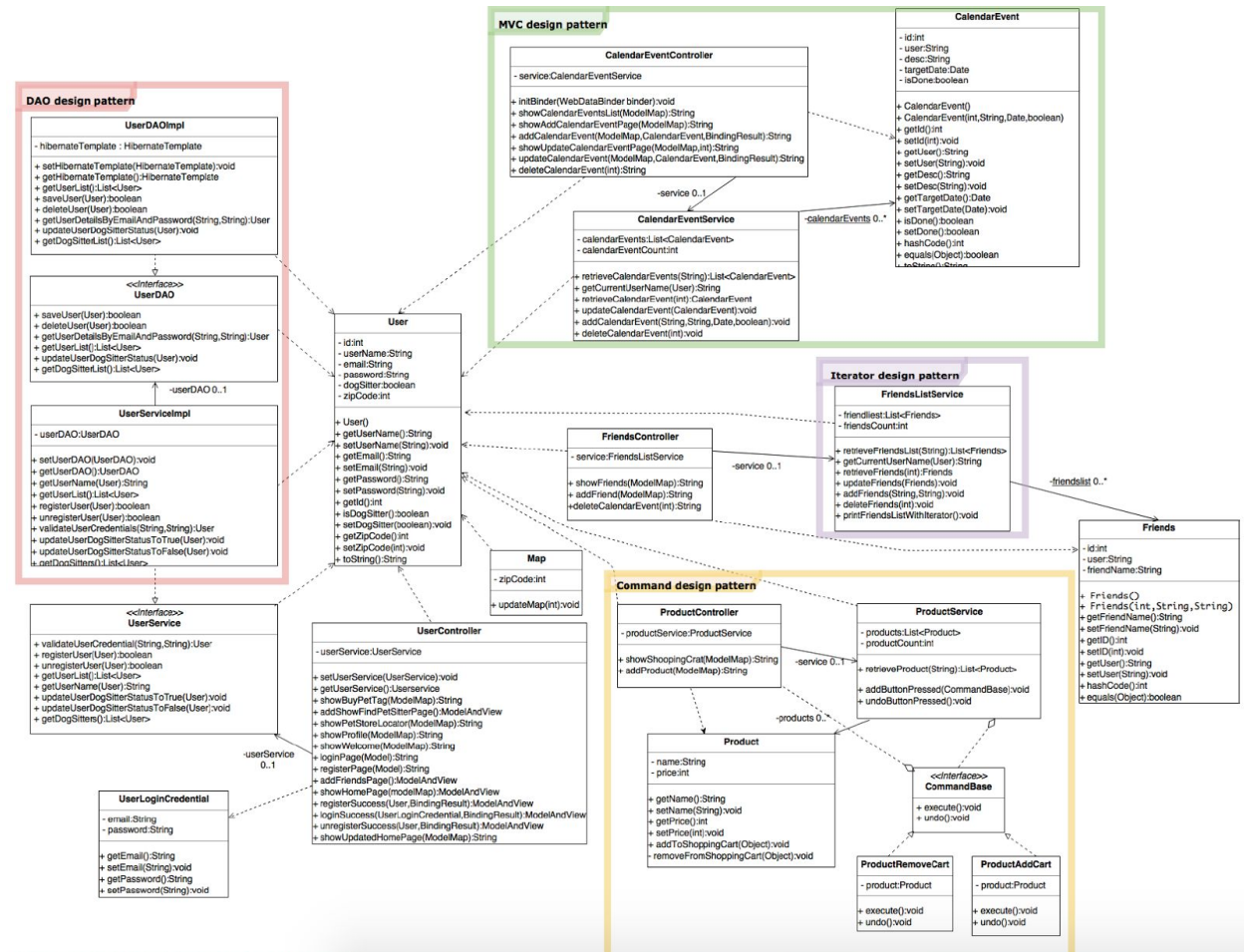2.) List the features that were not implemented from Part 2(table with ID and title).

US-03	As a user, I want to be able to log out.	Authentication	User	High
US-04	As a user, I want to be able to add photos of me and my pet.	Authentication	User	High
US-06	As a user, I want to be able to view other people's' profile and add them to my friend list.	Friends List	User	Medium
US-11	As a user, I want to be able to checkout.	Purchase	User	Medium
US-13	As a admin, I want have privilege to delete user's account.	Authentication	Admin	High
US-14	As a admin, I want to be able to add or remove products.	Management	Admin	Medium
US-17	As a user, I want to see pet store around me on a map.	Map	User	Medium
US-18	As a user, I want to be able to edit my profile (user's photo and descriptions).	Profile	User	Medium
US-21	As a user, I want to be able to email help desk.	Help Desk	User	Low

3.) Show your Part 2 class diagram and your final class diagram. What Changed? If it did not change much, then discuss how doing the design up front helped in the development.

Original Class Diagram:

Final Class Diagram:

The diagram contains the following classes:

User
- +firstName : String
- +lastName : String
- +userName : String
- -emailAddress : String
- -password : String
- +userID : Id
- +zip : int
- -createAccount(fn : firstName, ln : lastName, un : userName, em : emailAddress, pw : password, zp : zip): boolean
- -login(em : emailAddress, pw : password): void
- -logout(): void
- +getUserName(): String
- +deleteAccount(): boolean

Shopping Cart
- +items : int
- +totalPrice : int
- +setQuantity(): void
- +calPrice(): int

Product
- +name : String
- +price : int
- #addToShoppingCart(): void

Calendar
- +event_id : Id
- +event_name : String
- -addEvent(): void
- -deleteEvent(): void
- -editEvent(): void

user:Admin
- -deleteUser(): boolean
- -banUser(): boolean
- -addProduct(): boolean
- -removeProduct(): boolean

user:Account
- +address : String
- +city : String
- +State : String
- +country : String
- +phoneNumber : int
- +creditCardNumber : int
- +expirationDate : date
- +cvcNumber : int
- #buy()
- #setShippingAddress(fn : first name, ln : lastname, ad : address, ci : city, co : country, ph : phoneNumber): void
- #setBillingAddress(fn : first name, ln : lastname, ad : address, ci : city, co : country, ph : phoneNumber): void
- -setPaymentMethod(cc : creditCradNumber, ex : expirationDate, cvc : cvcNumber): void

Map
- +zip : int
- +changeZip(): void
- +updateMap(): void

Friends
- +friendPhoto : String
- +friendPetPhoto : String
- +yourFriend : boolean
- +addFriend(): void
- +removeFriend(): void
- +searchFriend(zip: int): ArrayList<Friends>
- +displayFriend(): void

user:account:DogSitter
- +editDogSitterInfo() void

user:account:DogOwner
- +userPhoto : String
- +petPhoto : String
- +userAboutMe : String
- -yourFriends : boolean
- +addUserPhoto(): boolean
- +removeUserPhoto(): boolean
- +addPetPhoto(): void
- +removePetPhoto(): void
- -editProfile(): void
- -viewYourFriends(): ArrayList<Friends>
- +searchForFriends(): ArrayList<Friends>
- +searchForDogSitter(): ArrayList<DogSitter>
- -updateDogSittingStatus(): void
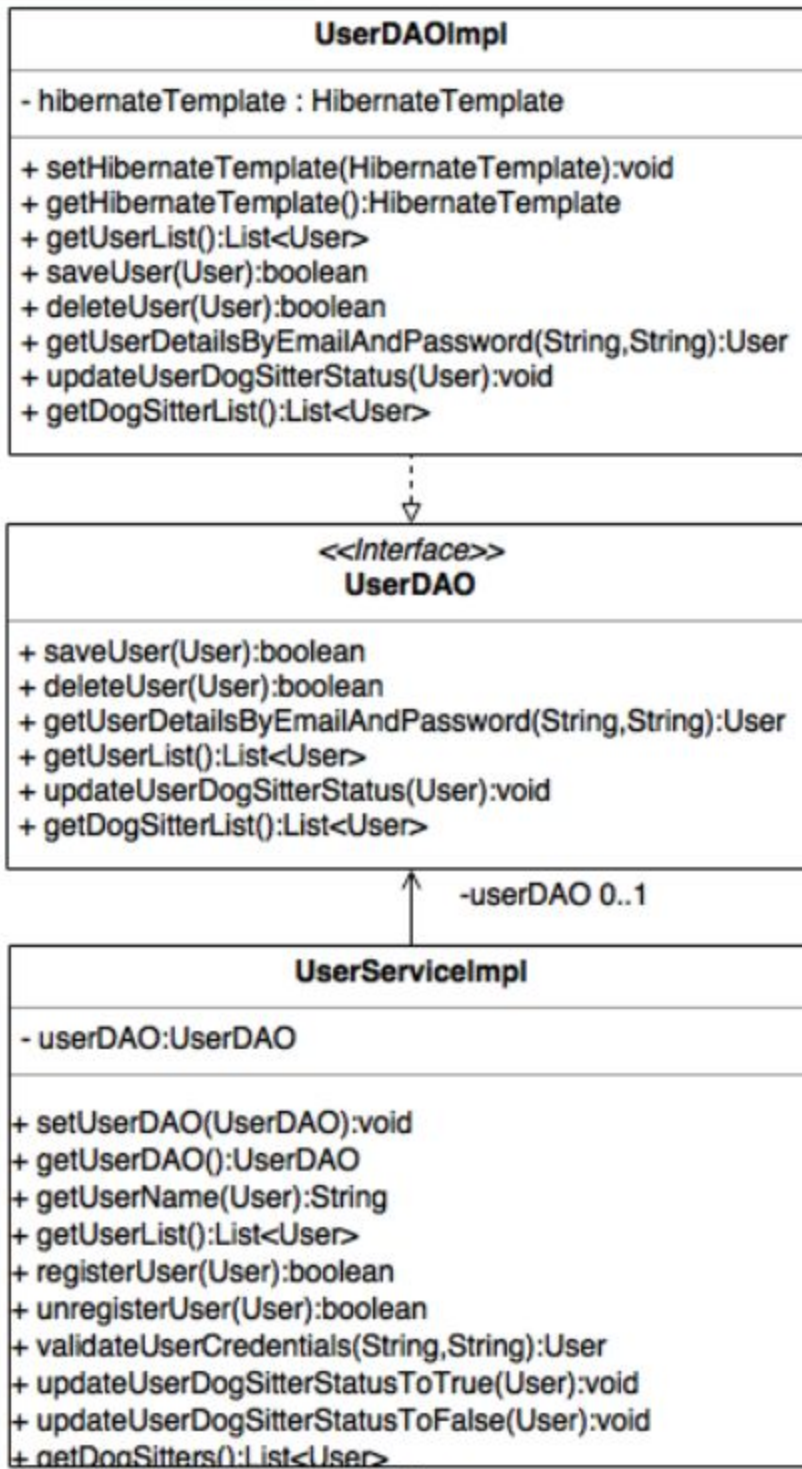- -deleteAccount(): boolean

When creating our original class diagram, we were unaware of how many design patterns we would be utilizing in our code. Originally we developed our class diagram without much thought how to incorporate design patterns. As we spent more time developing the architecture of our system, it was clear that many of our features could be implemented using a certain design pattern. For instance, When we

As you can see, in our final class diagram, we incorporated a few design patterns that made implementing certain features relatively easy.

4.) Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implements each design pattern (each design pattern as a separate image in the .PDF). If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram.

DAO design pattern:

DAO design pattern

UserDAOImpl

- hibernateTemplate : HibernateTemplate

+ setHibernateTemplate(HibernateTemplate):void
+ getHibernateTemplate():HibernateTemplate
+ getUserList():List<User>
+ saveUser(User):boolean
+ deleteUser(User):boolean
+ getUserDetailsByEmailAndPassword(String,String):User
+ updateUserDogSitterStatus(User):void
+ getDogSitterList():List<User>

<<Interface>>
UserDAO

+ saveUser(User):boolean
+ deleteUser(User):boolean
+ getUserDetailsByEmailAndPassword(String,String):User
+ getUserList():List<User>
+ updateUserDogSitterStatus(User):void
+ getDogSitterList():List<User>

-userDAO 0..1

UserServiceImpl

- userDAO:UserDAO

+ setUserDAO(UserDAO):void
+ getUserDAO():UserDAO
+ getUserName(User):String
+ getUserList():List<User>
+ registerUser(User):boolean
+ unregisterUser(User):boolean
+ validateUserCredentials(String,String):User
+ updateUserDogSitterStatusToTrue(User):void
+ updateUserDogSitterStatusToFalse(User):void
+ getDogSitters():List<User>

MVC Pattern:

MVC design pattern

CalendarEventController

- service:CalendarEventService

+ initBinder(WebDataBinder binder):void
+ showCalendarEventsList(ModelMap):String
+ showAddCalendarEventPage(ModelMap):String
+ addCalendarEvent(ModelMap,CalendarEvent,BindingResult):String
+ showUpdateCalendarEventPage(ModelMap,int):String
+ updateCalendarEvent(ModelMap,CalendarEvent,BindingResult):String
+ deleteCalendarEvent(int):String

-service 0..1

CalendarEventService

- calendarEvents:List<CalendarEvent>
- calendarEventCount:int

+ retrieveCalendarEvents(String):List<CalendarEvent>
+ getCurrentUserName(User):String
+ retrieveCalendarEvent(int):CalendarEvent
+ updateCalendarEvent(CalendarEvent):void
+ addCalendarEvent(String,String,Date,boolean):void
+ deleteCalendarEvent(int):void

-calendarEvents 0..*

CalendarEvent

- id:int
- user:String
- desc:String
- targetDate:Date
- isDone:boolean

+ CalendarEvent()
+ CalendarEvent(int,String,Date,boolean)
+ getId():int
+ setId(int):void
+ getUser():String
+ setUser(String):void
+ getDesc():String
+ setDesc(String):void
+ getTargetDate():Date
+ setTargetDate(Date):void
+ isDone():boolean
+ setDone():boolean
+ hashCode():int
+ equals(Object):boolean
+ toString():String
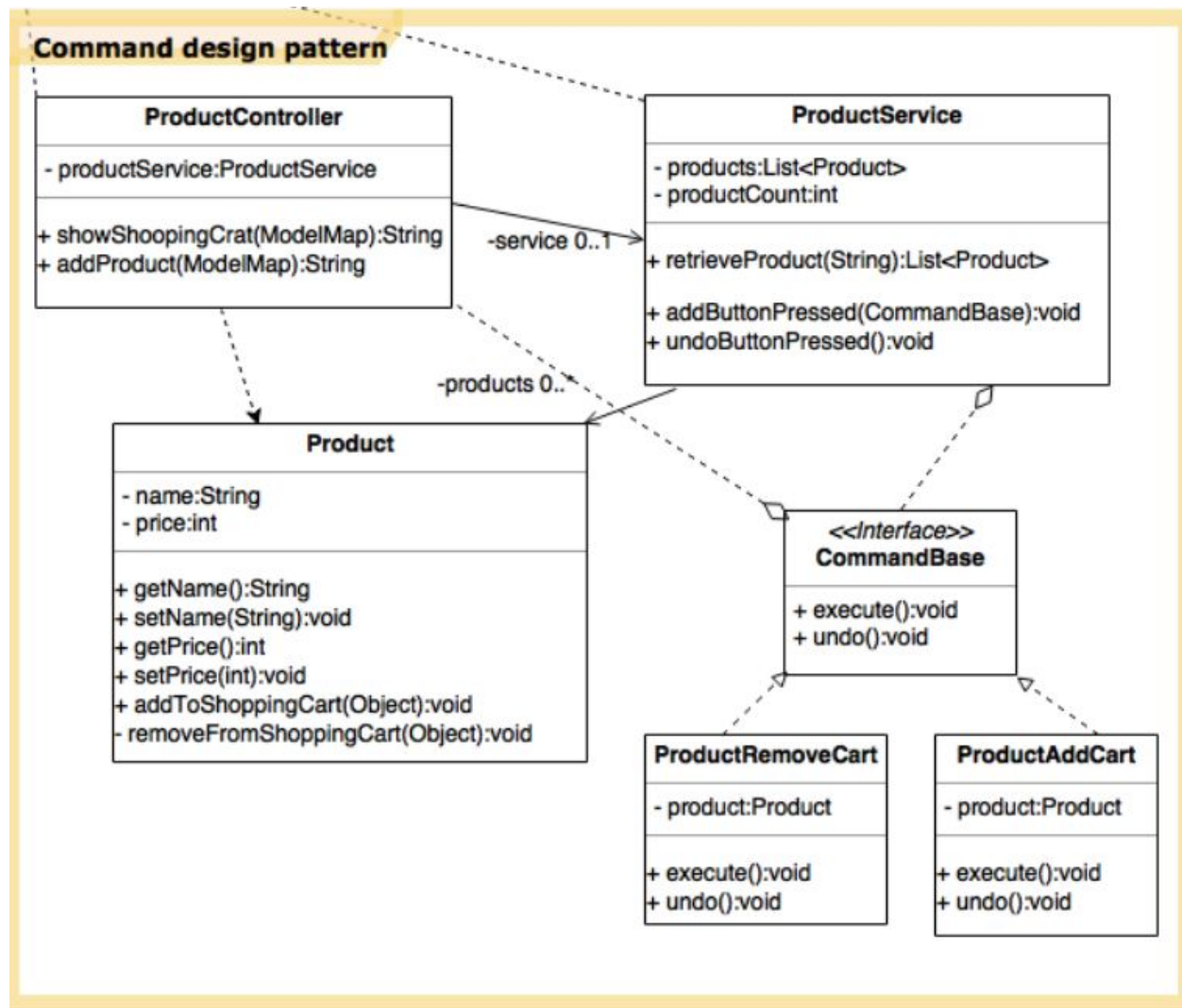
We used the MVC pattern to implement the User Interface of our system. The MVC pattern is a great tool for separating…

** architectural pattern consisting of three design patterns

Command Design Pattern:

Command design pattern

ProductController

- productService:ProductService

+ showShoopingCrat(ModelMap):String
+ addProduct(ModelMap):String

-service 0..1

ProductService

- products:List<Product>
- productCount:int

+ retrieveProduct(String):List<Product>

+ addButtonPressed(CommandBase):void
+ undoButtonPressed():void

-products 0..

Product

- name:String
- price:int

+ getName():String
+ setName(String):void
+ getPrice():int
+ setPrice(int):void
+ addToShoppingCart(Object):void
- removeFromShoppingCart(Object):void

<<Interface>>
CommandBase

+ execute():void
+ undo():void

ProductRemoveCart

- product:Product

+ execute():void
+ undo():void

ProductAddCart

- product:Product

+ execute():void
+ undo():void

In our app, we used the command pattern to implement the shopping cart feature. Utilizing the shopping cart feature

5.) What have you learned about the process of analysis and design now that you have stepped through the process to create, design, and implement a system?

Coming from a background of little to no experience in the process of analysis and design of a system, we as a team gained a very enriching experience. Designing a functional system in reality is no easy task, and requires a variety of different components. The process of designing the architecture of our system before actually coding it was a foreign concept to most of us, but in the end made the development process much easier. Beginning with Identifying the user features in our system made the transition to actually designing the class structure of our system fairly easy. We grasped an understanding of what features needed to be implemented

and we were able to design the class structure of our system . Before we even started to code, we had our class structure laid out including the design patterns we were going to utilize. In fact, we spent more time designing the architecture of our system on paper than we did actually coding it. Overall, this class gave us