Final Report, Rent-a-Pup:

Elias Ortiz, Callie Jones, Elijah Staple, Calvin Stoehr

1. Implemented Features

User Requirements	
ID	Title
UR-002	Renters can view and book puppies for rent
UR-003	Renters can provide feedback on their experience
UR-004	Renters can cancel bookings.

2. Not Implemented Features

Business Requirements	
ID	Title
BR-001	Employee payments are distributed on the first of each month
BR-002	Renters must provide valid payment upon placing puppy order
BR-003	Renters money is returned on a canceled booking.

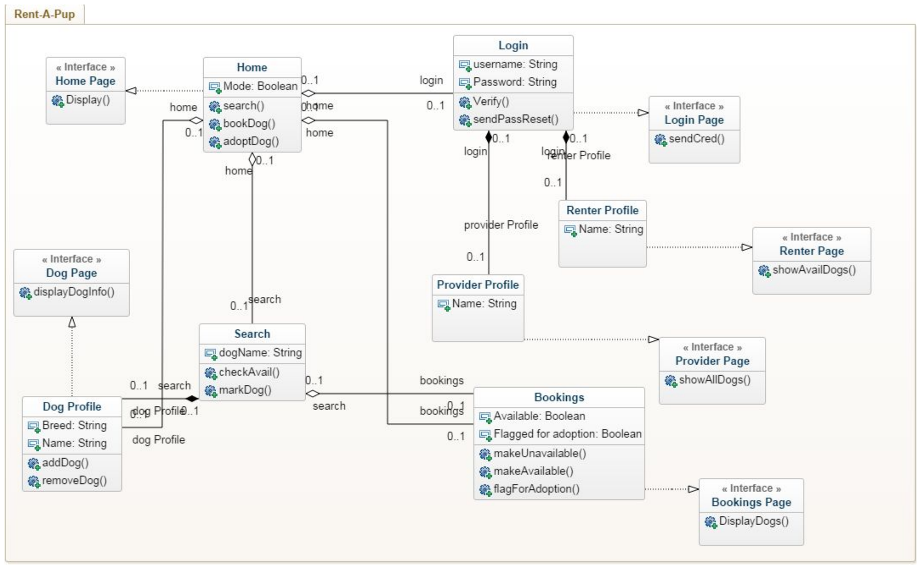
User Requirements	
ID	Title
UR-001	System admin can update available puppies for rent (Add/Remove)
UR-005	Admin can update puppy profiles
UR-006	Users can adopt puppies at the end of the reservation period

Non-Functional Requirements	
ID	Requirement
NFR-001	Website cannot be down on the weekends.
NFR-002	Website must provide secure system for payment information
NFR-003	Website must withstand heavy traffic

The reason most of the features remain unimplemented is because a majority of them are strictly related to the real-world application of this system. Because we currently have no intention of implementing this in the real world, we decided that these features weren't worth the time to create and flush out, and thought our time was better spent polishing up what we did complete.

3. Class Diagrams and Discussion on the Changes

Initial Diagram (for Project Part 2)



Project Part 3 Class Diagram :

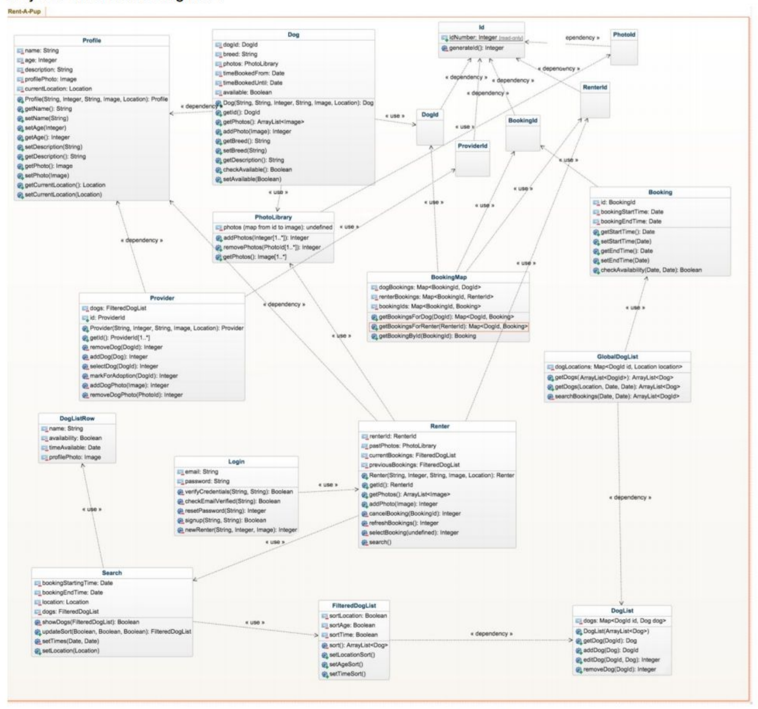

Rent-A-Pup
Profile
name: String
age: Integer
description: String
profilePhoto: Image
currentLocation: Location
Profile(String, Integer, String, Image, Location): Profile
getName(): String
setName(String)
setAge(Integer)
getAge(): Integer
setDescription(String)
getDescription(): String
getPhoto(): Image
setPhoto(Image)
getCurrentLocation(): Location
setCurrentLocation(Location)
Dog
dogId: DogId
breed: String
photos: PhotoLibrary
timeBookedFrom: Date
timeBookedUntil: Date
available: Boolean
Dog(String, String, Integer, String, Image, Location): Dog
getId(): DogId
getPhotos(): ArrayList<Image>
addPhoto(Image): Integer
getBreed(): String
setBreed(String)
getDescription(): String
checkAvailable(): Boolean
setAvailable(Boolean)
Id
idNumber: Integer {read-only}
generateId(): Integer
PhotoId
DogId
ProviderId
BookingId
RenterId
PhotoLibrary
photos (map from id to image): undefined
addPhotos(Integer[1..*]): Integer
removePhotos(PhotoId[1..*]): Integer
getPhotos(): Image[1..*]
Booking
id: BookingId
bookingStartTime: Date
bookingEndTime: Date
getStartTime(): Date
setStartTime(Date)
getEndTime(): Date
setEndTime(Date)
checkAvailability(Date, Date): Boolean
BookingMap
dogBookings: Map<BookingId, DogId>
renterBookings: Map<BookingId, RenterId>
bookingIds: Map<BookingId, Booking>
getBookingsForDog(DogId): Map<DogId, Booking>
getBookingsForRenter(RenterId): Map<DogId, Booking>
getBookingById(BookingId): Booking
Provider
dogs: FilteredDogList
id: ProviderId
Provider(String, Integer, String, Image, Location): Provider
getId(): ProviderId[1..*]
removeDog(DogId): Integer
addDog(Dog): Integer
selectDog(DogId): Integer
markForAdoption(DogId): Integer
addDogPhoto(Image): Integer
removeDogPhoto(PhotoId): Integer
GlobalDogList
dogLocations: Map<DogId id, Location location>
getDogs(ArrayList<DogId>): ArrayList<Dog>
getDogs(Location, Date, Date): ArrayList<Dog>
searchBookings(Date, Date): ArrayList<DogId>
DogListRow
name: String
availability: Boolean
timeAvailable: Date
profilePhoto: Image
Login
email: String
password: String
verifyCredentials(String, String): Boolean
checkEmailVerified(String): Boolean
resetPassword(String): Integer
signup(String, String): Boolean
newRenter(String, Integer, Image): Integer
Renter
renterId: RenterId
pastPhotos: PhotoLibrary
currentBookings: FilteredDogList
previousBookings: FilteredDogList
Renter(String, Integer, String, Image, Location): Renter
getId(): RenterId
getPhotos(): ArrayList<Image>
addPhoto(Image): Integer
cancelBooking(BookingId): Integer
refreshBookings(): Integer
selectBooking(undefined): Integer
search()
Search
bookingStartingTime: Date
bookingEndTime: Date
location: Location
dogs: FilteredDogList
showDogs(FilteredDogList): Boolean
updateSort(Boolean, Boolean, Boolean): FilteredDogList
setTimes(Date, Date)
setLocation(Location)
FilteredDogList
sortLocation: Boolean
sortAge: Boolean
sortTime: Boolean
sort(): ArrayList<Dog>
setLocationSort()
setAgeSort()
setTimeSort()
DogList
dogs: Map<DogId id, Dog dog>
DogList(ArrayList<Dog>)
getDog(DogId): Dog
addDog(Dog): DogId
editDog(DogId, Dog): Integer
removeDog(DogId): Integer
« use »
« dependency »

*diagram is missing single dotted <<use>> arrow from Provider to PhotoLibrary

The second picture is our most recent class diagram, and there are still things that we would change, especially if this system was going to be implemented in real life. Many of our “use” arrows could actually be association arrows, but that would depend on tweaks that would happen with a more real life focused implementation. Additionally, all of the dependency arrows between generic id and the specific id’s are actually generalization arrows.

4. Design Patterns implemented in our Prototype.

- Template
    - “Profile” superclass allows for significant code reuse via inheritance in the form of a template.
    - We did this so that we could have very easy creation and integration of future profile types.
- Command
    - cancelBooking() undoes what addBooking() did, following the command design pattern.
    - We used this to help streamline our database queries, because most of them would be of the form addBooking() or cancelBooking().
- Observer
    - We create and then use filtered and then sorted lists of dogs many times in the project.
    - Utilizing the Observer design pattern was simply the most efficient way of accomplishing that.

5. Things we’ve learned about the Analysis and Design process.

- Having short term goals along with long term goals makes the process far more palatable, without the checkpoints on this project it would not be nearly as complete as it is now.
- Having a solid plan going into the first design phase saves a lot of time on numerous “time-wasters” ie. consistently naming functions, exact responsibilities of functions, interactions between objects and functions, etc.
- Regular Meetings with all group members helps to keep everyone on the same page, and also helps minimize how much time anyone could be stuck on something or waiting on someone else to finish a dependancy.