Rental App

Emily Reynolds

Table of Contents

Problem statement: When people fly to new cities, they often rent vehicles. These vehicles are usually undesirable, and the end-user does not have a choice as to the specific vehicle they receive. Users are even subjected to vehicle-class changes without notice after booking due to usage. The only options that allow for users to pick out a specific vehicle are peer-to-peer.

Objectives of the system: This system should help travelers to rent a more desirable vehicle, with finite control over which exact vehicle is rented. The rental company will be able to charge higher rates for the same vehicle by giving users more control over the transaction. Specific Features:

Typical customers:

- Airport travelers
- Road trip travelers
- Rideshare operators
- Rental companies
- Travel agencies

Project planning and development approach:

1. Software:
    a. Front-end: React
    b. Runtime: Node.js
    c. Back-end: Kotlin
    d. Database: MongoDB 2. Hardware:
    a. PC
    b. Servers 3. Network:
    a. No speed requirements

Specific Features:

1. End users can view and rent specific vehicles
2. Rental Companies can manage cars in the system including:
    a. Location
    b. Maintenance
3. Rental companies can track users including:
    a. Insurance
    b. Travel history
    c. Vehicle preferences
    d. Vehicle state at time of return (aggregate renter score)

Emily Reynolds 2/3/23

Problem Statement: When people fly to new cities, they often rent vehicles. These vehicles are usually undesirable, and the end-user does not have a choice as to the specific vehicle they receive. Users are even subjected to vehicle-class changes without notice after booking due to usage. The only options that allow for users to pick out a specific vehicle are peer-to-peer services without the trust conferred by large rental corporations.

Glossary of Terms:

Vehicle class – classification of vehicles including classifications like luxury car, mid-size sedan, SUV, and Pickup truck

Reservation – a claim for a specific vehicle for a specific amount of time starting at a specific location and ending at a specific location **System Requirements:**

No.	Priority	Description
REQ-1	High	Customers can view and select vehicle inventory at their specific pickup location.
REQ-2	High	Customers can return to any location, but must be selected at time of rental
REQ-3	High	Service is available via webapp
REQ-4	High	Customer can select specific vehicle three to five days before rental. Same-day rentals are not guaranteed.
REQ-5	High	Rental company employees must be able to manage location of vehicles and details of customer rentals via an admin-panel
REQ-6	Medium	Cost is determined by pickup location, dropoff location, vehicle, and duration of rental
REQ-7	Medium	Payment must be made using credit or debit
REQ-8	Medium	Additional services are offered before checkout like refueling, toll pass, and insurance
NF-REQ-1	High	Should be able to be hosted on minimal server hardware
NF-REQ-2	Medium	Should have a 99.9% uptime
NF-REQ-3	Medium	Should have modern webapp appearance that is simple to navigate for all levels of users
NF-REQ-4	Medium	Should not visibly lag upon loading new vehicles with reasonable internet connection
NF-REQ-5	Medium	Should be easily configured and support different business models in different locations and for different clients
UI-REQ-1	High	Must have legible design with clearly defined features
UI-REQ-2	Medium	Should be attractive, at least not off-putting to look at
UI-REQ-3	Medium	UI should utilize a simple one-column design with a navbar at the top
UI-REQ-4	Medium	There should be pictures of the specific car that is being offered rather than stock images

UI-REQ-5	Medium	User should be able to rate vehicles and view previous rentals

Rental Company App

Key Stakeholders:

- Rental company employees
- People who rent cars
- Rental company admins

Actors and goals:

- Primary actors o Customer: This actor can reserve a specific vehicle from a specific location and pay for it upon receipt of the vehicle.
    - Rental company employee: This actor can track vehicles, update history including maintenance, modify reservations, and dispense keys to customers along with receiving payment.
- Secondary actors:
    - Rental company admin: This actor can add or decommission vehicles, set prices, add or remove store locations, and any task available to employees.
    - System: This actor is responsible for maintaining accuracy of vehicle locations and allowing for customers to easily reserve their desired cars.

Use Cases:

Rental company admin (18) :

- Add or remove vehicle (2)
- Add or remove employee (2)
- Add or remove location (2)
- Add or modify pricing (2)
- View Account info (2)
- View Vehicle history (4)

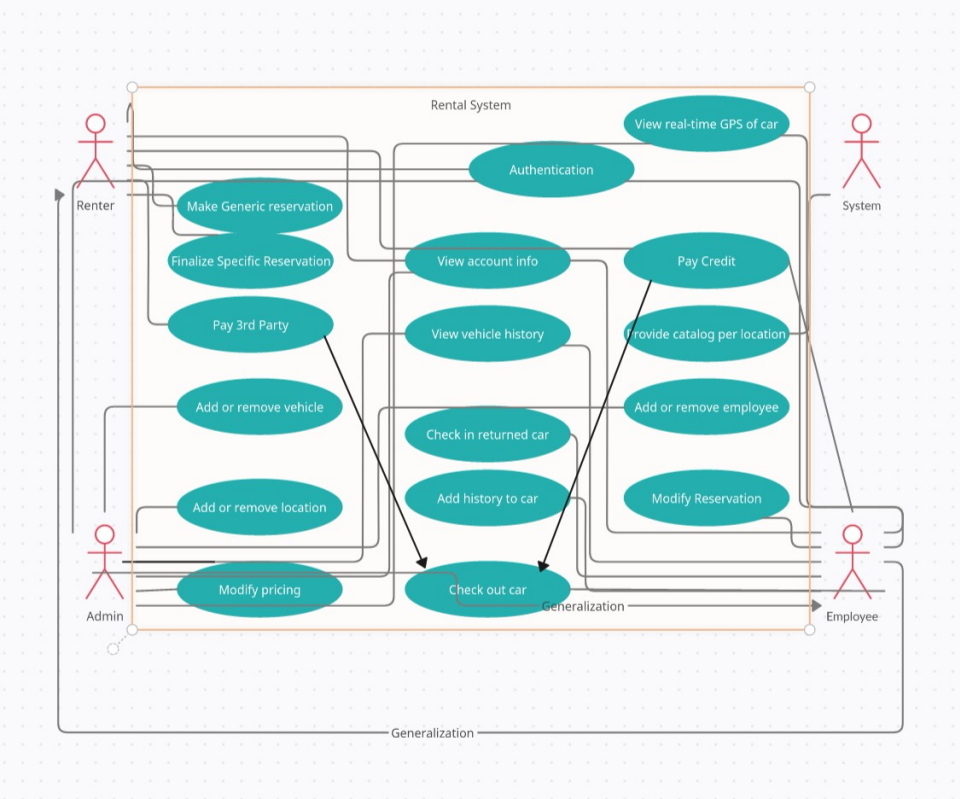
Customer (12) :

- Make generic reservation at a specific location (2)
- Make specific reservation within specified timeframe (3)
- Pay for vehicle using credit/debit card (2)
- Pay for vehicle using proprietary payment systems (Apple, Samsung, Google) (5)

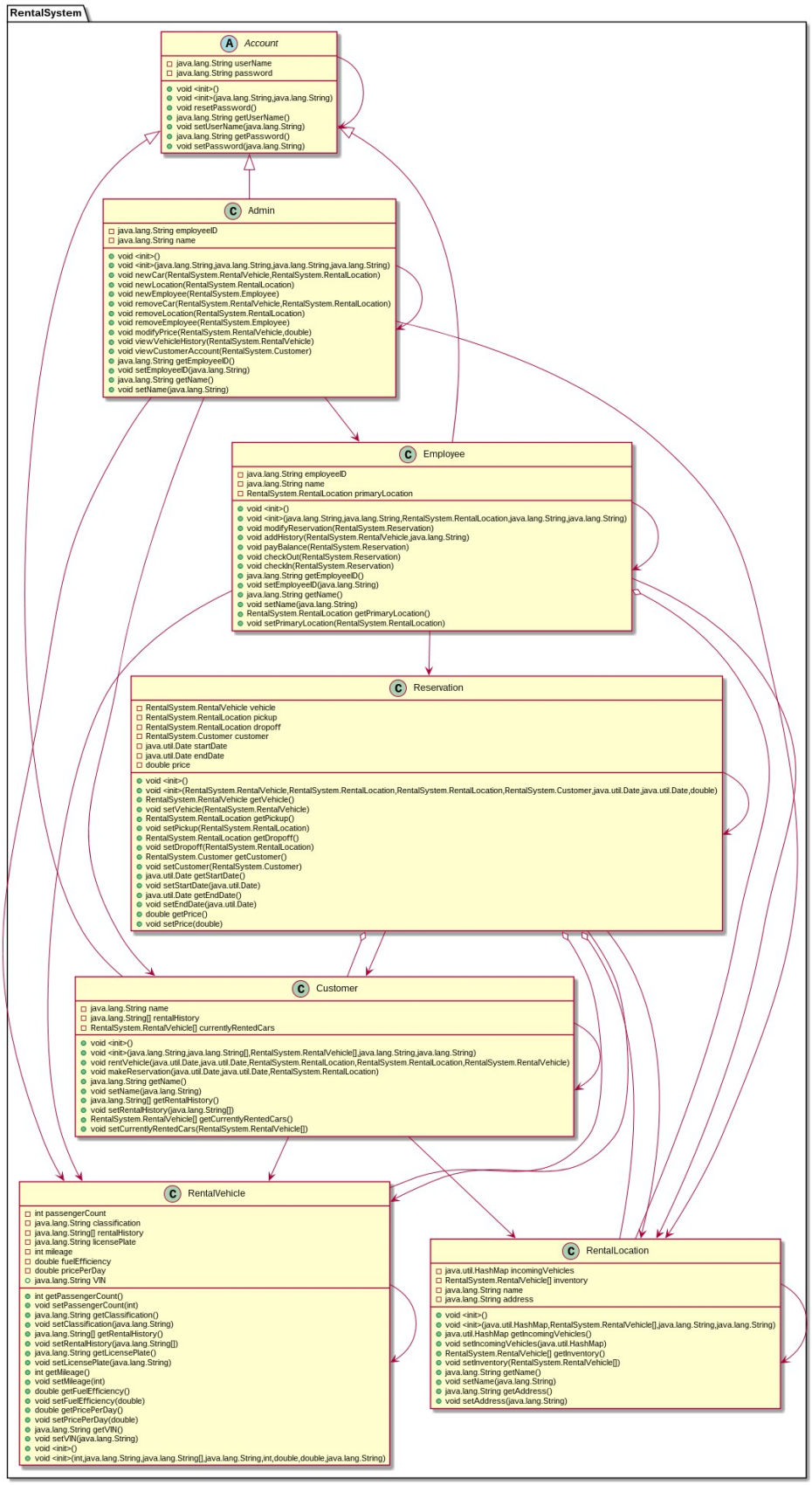
Rental company employee (10) :

- Accept receipt of returned vehicle (2)
- Modify history of vehicle (2)
- Distribute keys for rental (1)
- View real-time location of vehicle (2)
- Modify reservation (2)
- Accept customer payment via credit card (1) System (2) :

- Providing catalogs of vehicles by location (2)
- Authentication (4)



Class diagrams:


RentalSystem
Account
java.lang.String userName
java.lang.String password
void <init>()
void <init>(java.lang.String,java.lang.String)
void resetPassword()
java.lang.String getUserName()
void setUserName(java.lang.String)
java.lang.String getPassword()
void setPassword(java.lang.String)
Admin
java.lang.String employeeID
java.lang.String name
void <init>()
void <init>(java.lang.String,java.lang.String,java.lang.String,java.lang.String)
void newCar(RentalSystem.RentalVehicle,RentalSystem.RentalLocation)
void newLocation(RentalSystem.RentalLocation)
void newEmployee(RentalSystem.Employee)
void removeCar(RentalSystem.RentalVehicle,RentalSystem.RentalLocation)
void removeLocation(RentalSystem.RentalLocation)
void removeEmployee(RentalSystem.Employee)
void modifyPrice(RentalSystem.RentalVehicle,double)
void viewVehicleHistory(RentalSystem.RentalVehicle)
void viewCustomerAccount(RentalSystem.Customer)
java.lang.String getEmployeeID()
void setEmployeeID(java.lang.String)
java.lang.String getName()
void setName(java.lang.String)
Employee
java.lang.String employeeID
java.lang.String name
RentalSystem.RentalLocation primaryLocation
void <init>()
void <init>(java.lang.String,java.lang.String,RentalSystem.RentalLocation,java.lang.String,java.lang.String)
void modifyReservation(RentalSystem.Reservation)
void addHistory(RentalSystem.RentalVehicle,java.lang.String)
void payBalance(RentalSystem.Reservation)
void checkOut(RentalSystem.Reservation)
void checkin(RentalSystem.Reservation)
java.lang.String getEmployeeID()
void setEmployeeID(java.lang.String)
java.lang.String getName()
void setName(java.lang.String)
RentalSystem.RentalLocation getPrimaryLocation()
void setPrimaryLocation(RentalSystem.RentalLocation)
Reservation
RentalSystem.RentalVehicle vehicle
RentalSystem.RentalLocation pickup
RentalSystem.RentalLocation dropoff
RentalSystem.Customer customer
java.util.Date startDate
java.util.Date endDate
double price
void <init>()
void <init>(RentalSystem.RentalVehicle,RentalSystem.RentalLocation,RentalSystem.RentalLocation,RentalSystem.Customer,java.util.Date,java.util.Date,double)
RentalSystem.RentalVehicle getVehicle()
void setVehicle(RentalSystem.RentalVehicle)
RentalSystem.RentalLocation getPickup()
void setPickup(RentalSystem.RentalLocation)
RentalSystem.RentalLocation getDropoff()
void setDropoff(RentalSystem.RentalLocation)
RentalSystem.Customer getCustomer()
void setCustomer(RentalSystem.Customer)
java.util.Date getStartDate()
void setStartDate(java.util.Date)
java.util.Date getEndDate()
void setEndDate(java.util.Date)
double getPrice()
void setPrice(double)
Customer
java.lang.String name
java.lang.String[] rentalHistory
RentalSystem.RentalVehicle[] currentlyRentedCars
void <init>()
void <init>(java.lang.String,java.lang.String[],RentalSystem.RentalVehicle[],java.lang.String,java.lang.String)
void rentVehicle(java.util.Date,java.util.Date,RentalSystem.RentalLocation,RentalSystem.RentalLocation,RentalSystem.RentalVehicle)
void makeReservation(java.util.Date,java.util.Date,RentalSystem.RentalLocation)
java.lang.String getName()
void setName(java.lang.String)
java.lang.String[] getRentalHistory()
void setRentalHistory(java.lang.String[])
RentalSystem.RentalVehicle[] getCurrentlyRentedCars()
void setCurrentlyRentedCars(RentalSystem.RentalVehicle[])
RentalVehicle
int passengerCount
java.lang.String classification
java.lang.String[] rentalHistory
java.lang.String licensePlate
int mileage
double fuelEfficiency
double pricePerDay
java.lang.String VIN
int getPassengerCount()
void setPassengerCount(int)
java.lang.String getClassification()
void setClassification(java.lang.String)
java.lang.String[] getRentalHistory()
void setRentalHistory(java.lang.String[])
java.lang.String getLicensePlate()
void setLicensePlate(java.lang.String)
int getMileage()
void setMileage(int)
double getFuelEfficiency()
void setFuelEfficiency(double)
double getPricePerDay()
void setPricePerDay(double)
java.lang.String getVIN()
void setVIN(java.lang.String)
void <init>()
void <init>(int,java.lang.String,java.lang.String[],java.lang.String,int,double,double,java.lang.String)
RentalLocation
java.util.HashMap incomingVehicles
RentalSystem.RentalVehicle[] inventory
java.lang.String name
java.lang.String address
void <init>()
void <init>(java.util.HashMap,RentalSystem.RentalVehicle[],java.lang.String,java.lang.String)
java.util.HashMap getIncomingVehicles()
void setIncomingVehicles(java.util.HashMap)
RentalSystem.RentalVehicle[] getInventory()
void setInventory(RentalSystem.RentalVehicle[])
java.lang.String getName()
void setName(java.lang.String)
java.lang.String getAddress()
void setAddress(java.lang.String)

RentalVehicle:

The vehicle class is primarily an abstraction that contains vital information about the rental vehicle in question. It represents the vehicle's classification as a string. Individual classifications are only useful for sorting. Other vital stats include VIN, license plate, mileage, how many seats, and fuel economy.

RentalLocation:

The RentalLocation is any physical location that a rental can be checked in or out from. They have a list of in stock vehicles as well as a map containing vehicles that are scheduled to arrive soon and their projected arrival.

Reservation:

Reservations are accumulator classes that have pickup and dropoff date and location as well as a vehicle and customer account.

Account:

Account is an abstract class that contains only a username and password. It is extended as Customer, Admin, and Employee.

Customer:

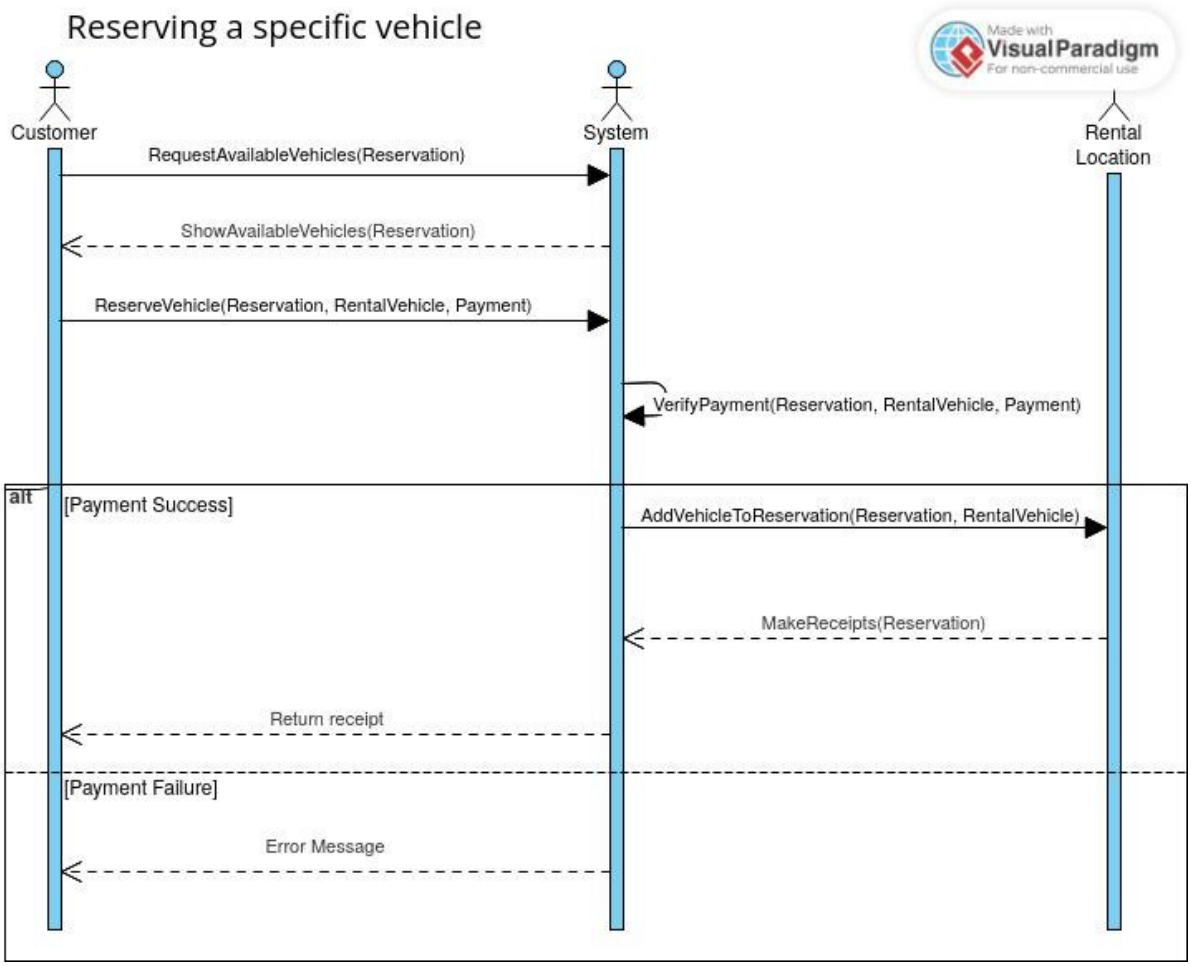
Customers have a history, list of currently rented cars. They can rent RentalVehicles.

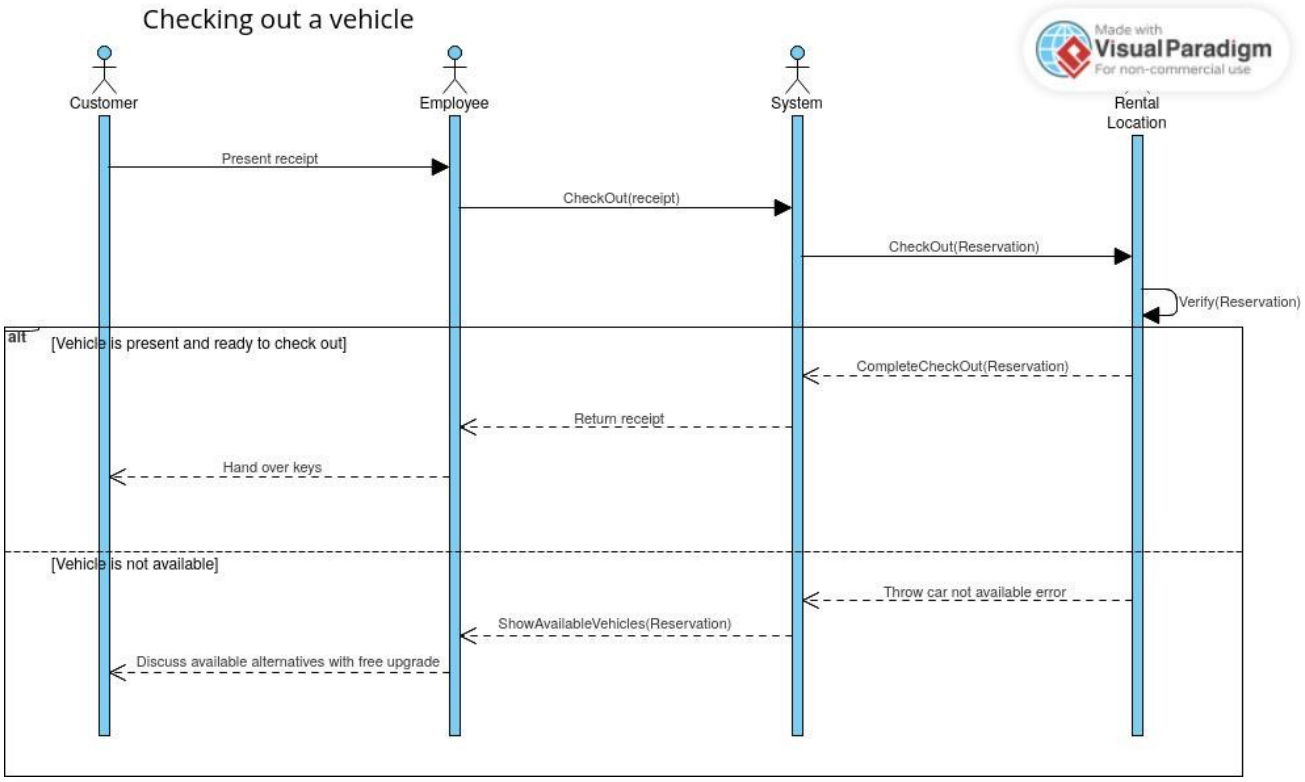
Employee:

Employees have employee id numbers, names, and a primary working location. They have the ability to modify reservations, check out and check in vehicles, and add history to vehicles.

Admin:

Admins have employee id numbers and names. They are able to create and destroy vehicles, locations, and employees.


Reserving a specific vehicle
Made with VisualParadigm For non-commercial use
Customer
System
Rental Location
RequestAvailableVehicles(Reservation)
ShowAvailableVehicles(Reservation)
ReserveVehicle(Reservation, RentalVehicle, Payment)
VerifyPayment(Reservation, RentalVehicle, Payment)
alt
[Payment Success]
AddVehicleToReservation(Reservation, RentalVehicle)
MakeReceipts(Reservation)
Return receipt
[Payment Failure]
Error Message

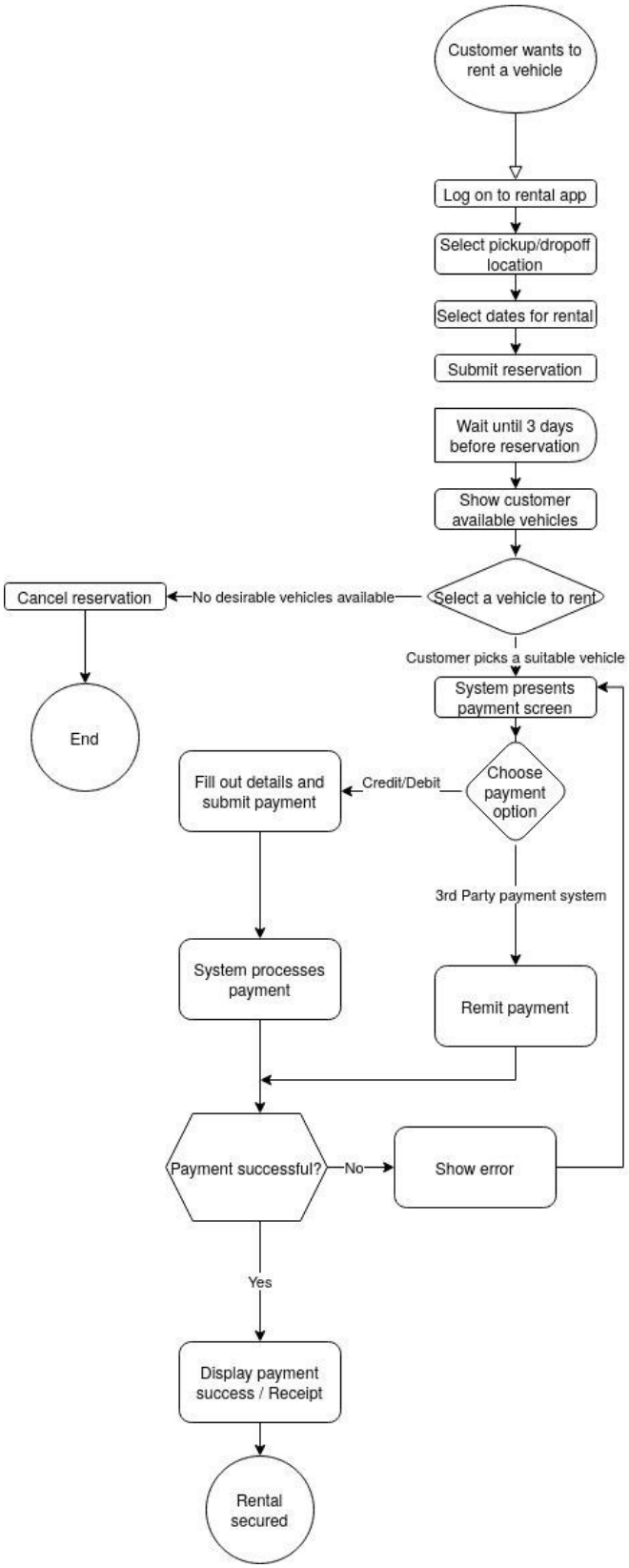


Use Case: Customer reserving a vehicle

Actions: Customer selects dates and locations for rental. Once selected, customer waits until the rental period (3 days prior). Customer is given a choice of vehicles. If none are desirable the customer can end process here. Otherwise, the customer selects a vehicle and pays for it.

States:

- Start: Customer wants a vehicle
- Final: No desirable car available
- Final: Car rented

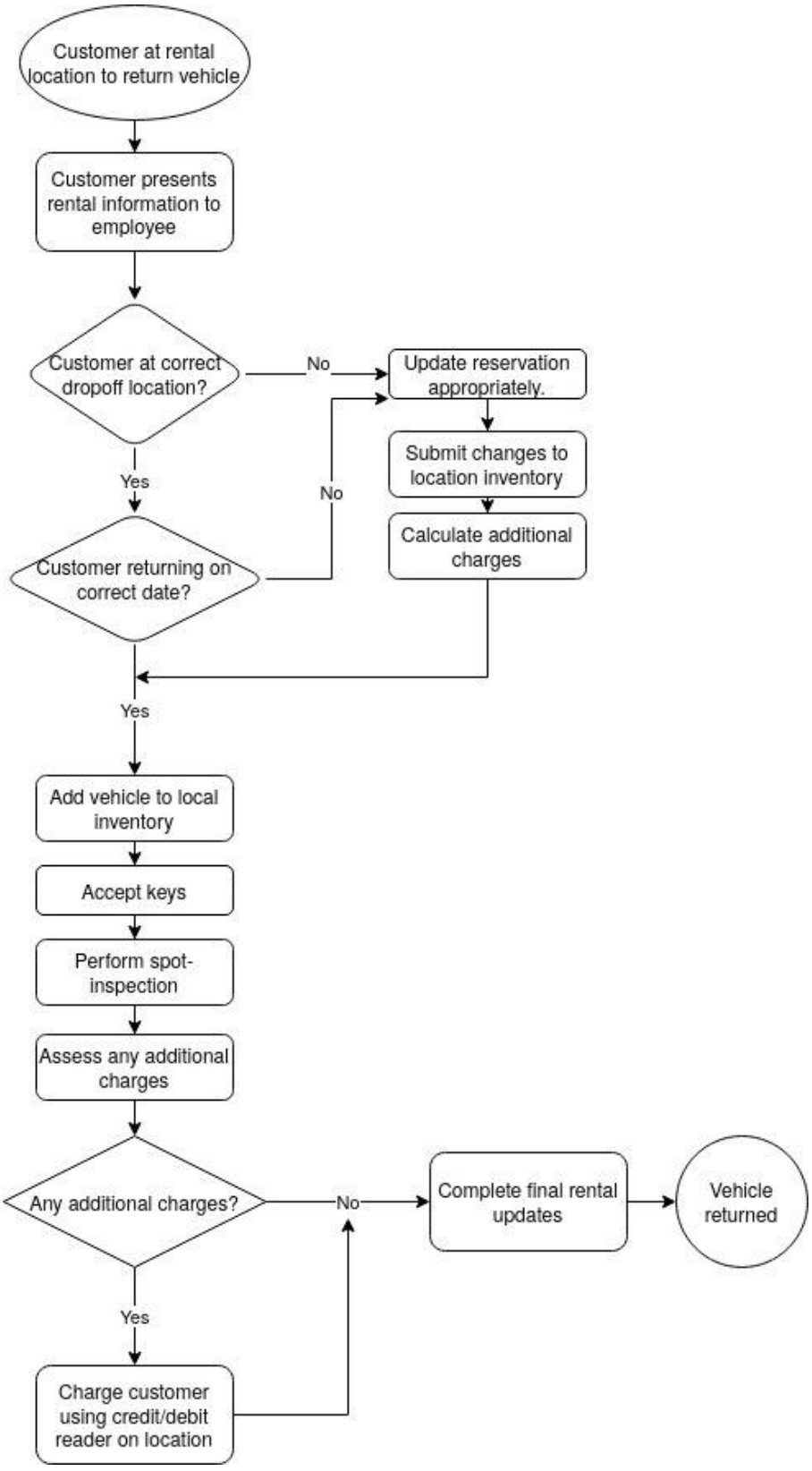

Customer wants to rent a vehicle
Log on to rental app
Select pickup/dropoff location
Select dates for rental
Submit reservation
Wait until 3 days before reservation
Show customer available vehicles
Select a vehicle to rent
Cancel reservation
No desirable vehicles available
End
Customer picks a suitable vehicle
System presents payment screen
Choose payment option
Credit/Debit
Fill out details and submit payment
3rd Party payment system
System processes payment
Remit payment
Payment successful?
No
Show error
Yes
Display payment success / Receipt
Rental secured

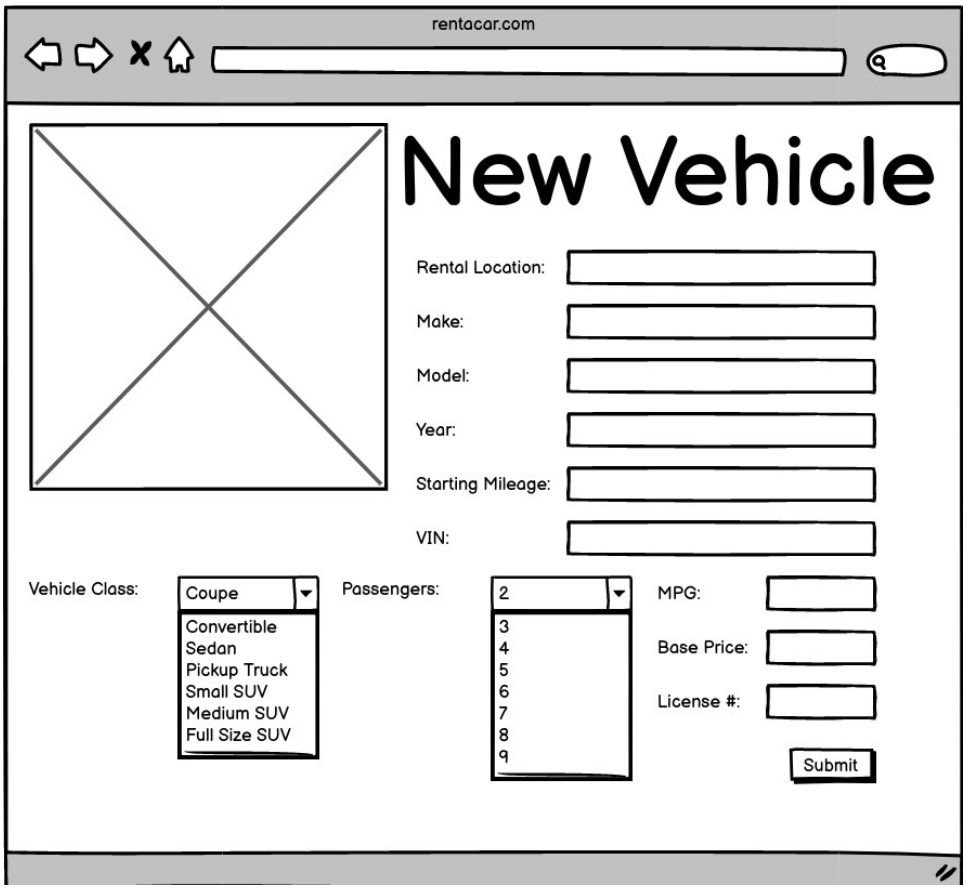
Use Case: Customer returning a vehicle

Actions: Customer arrives at a rental location to drop off car. Employee rectifies rental if location or enddate have changed from original reservation. If there is an additional charge for reservation changes, it is tabulated. The employee takes keys and does a spot inspection. If anything is noticeably damaged, additional charges are tabulated. All additional charges are presented to customer and paid. Return is completed

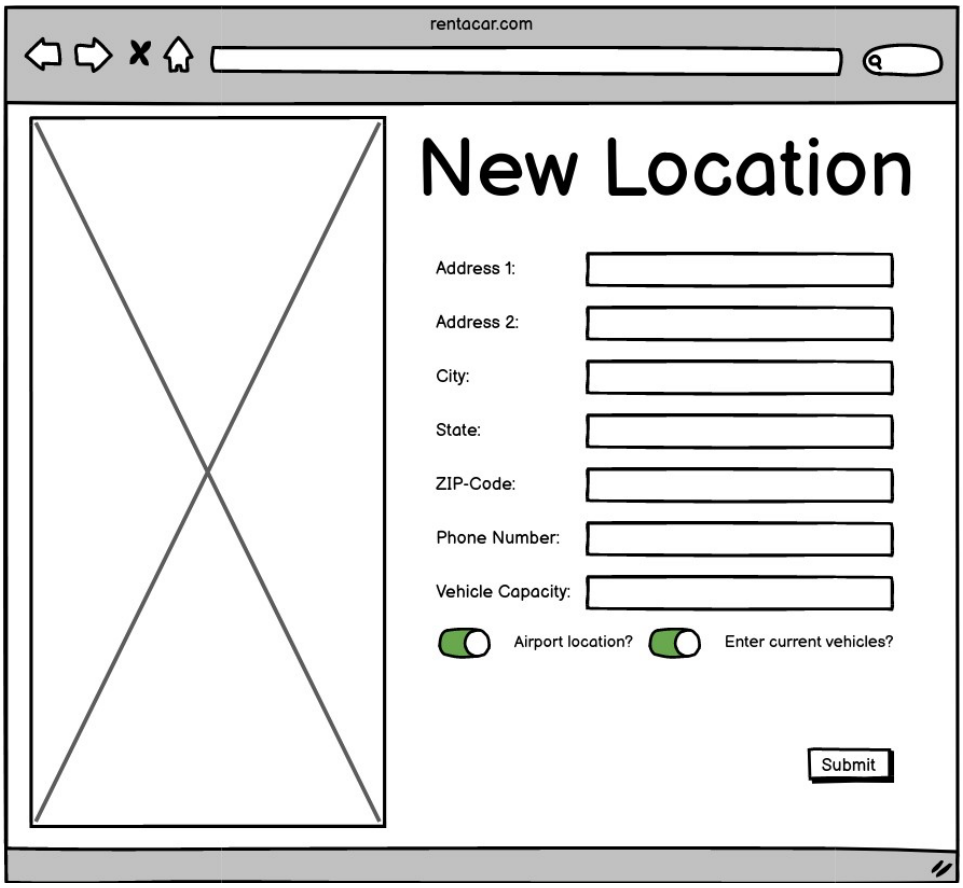
States:

- Start: Customer arrives at a dropoff location
- Final: Car returned

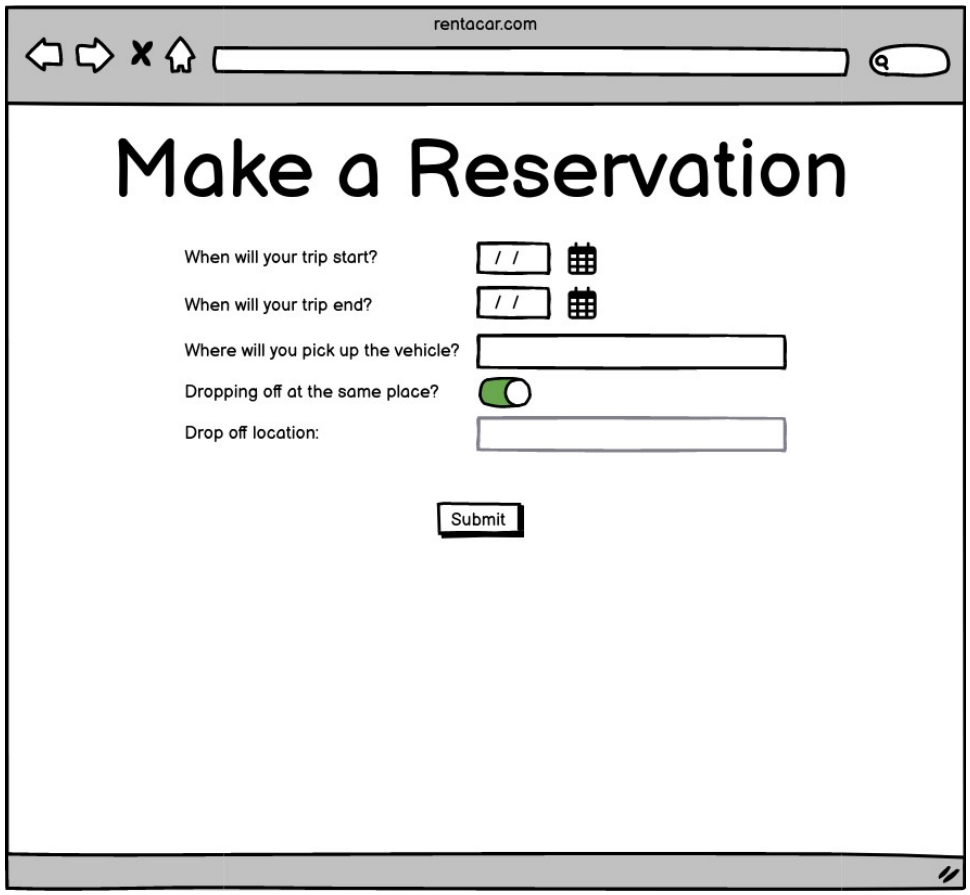

Customer at rental location to return vehicle
Customer presents rental information to employee
Customer at correct dropoff location?
No
Update reservation appropriately.
Submit changes to location inventory
Calculate additional charges
Yes
No
Customer returning on correct date?
Yes
Add vehicle to local inventory
Accept keys
Perform spot-inspection
Assess any additional charges
Any additional charges?
No
Complete final rental updates
Vehicle returned
Yes
Charge customer using credit/debit reader on location



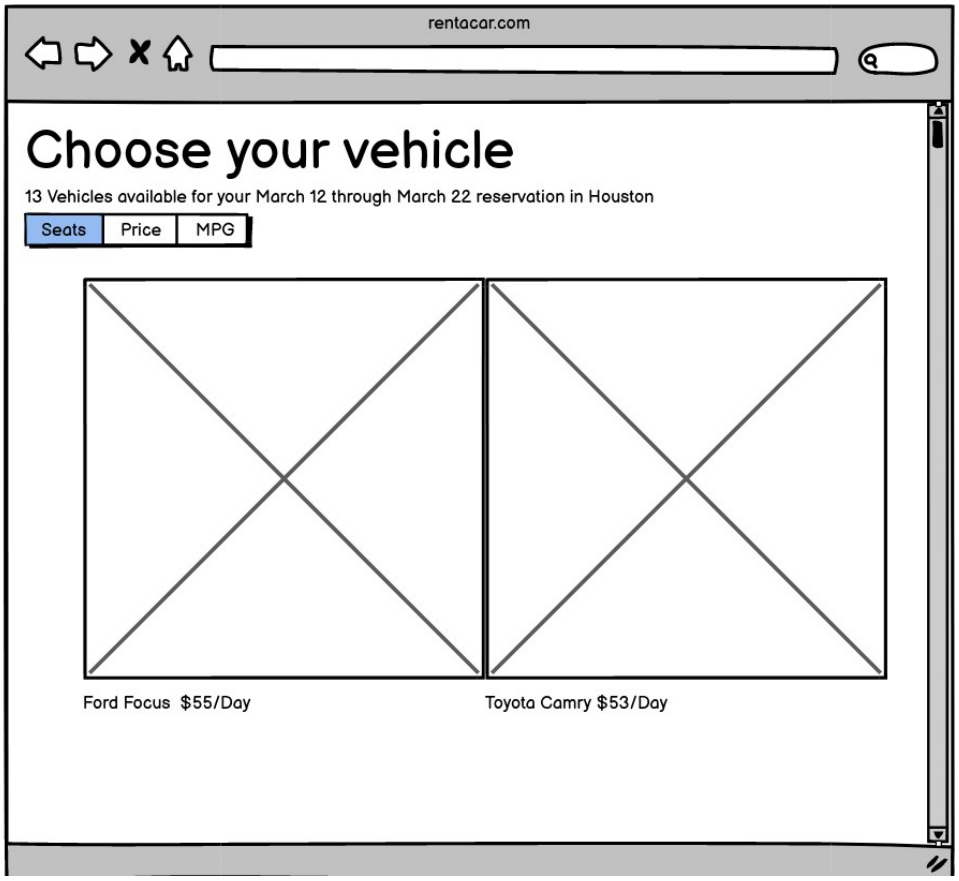
Creating a new vehicle is done by a corporate employee. The portal seen above is a one page app. It requires end-users to interact with 9 text fields, 2 drop-down menus, and then submit. The rental location must be verified to exist. 12 distinct interactions.



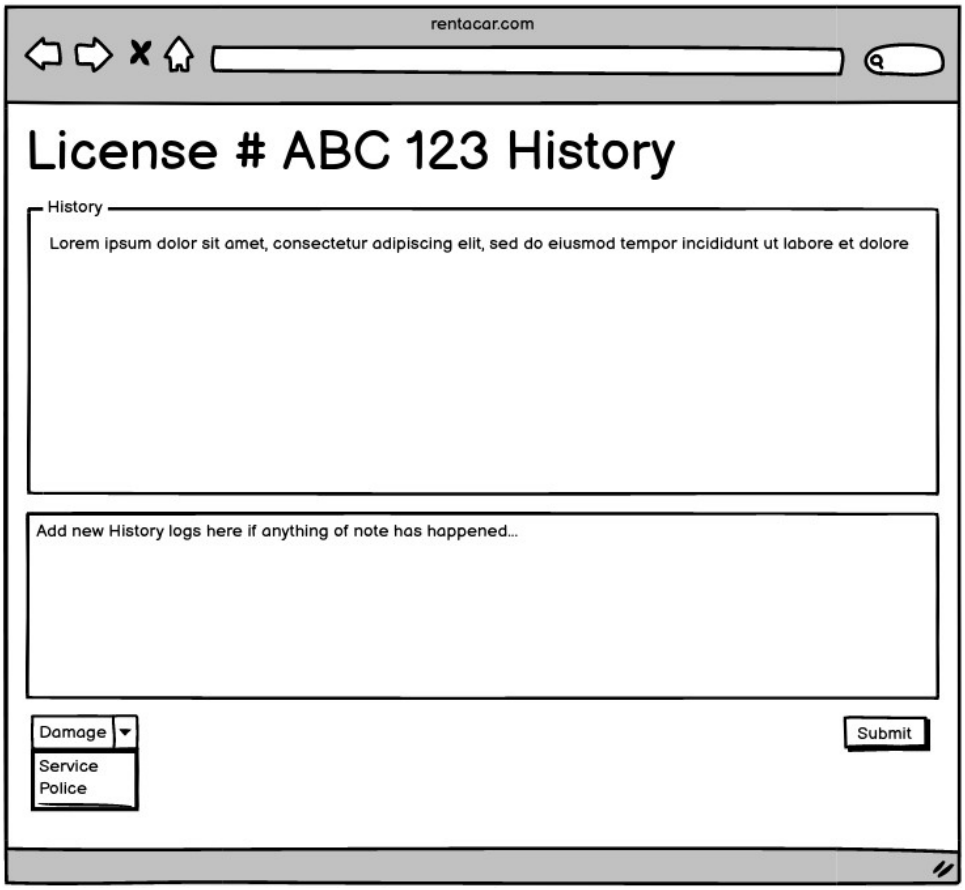
Adding a new location is also done by a corporate employee. The Employee must add all address information, a phone number, and vehicle capacity. There are two on/off switches. One indicates whether or not the location is at an airport. The other indicates whether or not the employee will immediately be adding vehicles to the location. If this is the case, a the user will immediately be taken to the new vehicle screen after hitting submit. 10 distinct interactions.



Making a generic interaction is the first time a customer will interact with the system. The User will have to enter a start and end date for the rental and the pickup location. If the user then selects that they will drop off at the same location, they will be able to submit immediately. If, however they want to drop off the vehicle elsewhere, they will need to enter that information. 6 distinct interactions.



When a customer is allowed to select their rental vehicle, they will be presented with a gallery similar to Google images. They can scroll through images, sort them by seat number, price, or mpg. Upon clicking on a vehicle users will be taken to a portal similar to creating the vehicle (done by admins) but all of the text is static. The amount of times a user interacts with the system is dependent upon how quickly they find a desirable vehicle. After selecting a vehicle the user would then go through the payment process. >= 3 distinct interactions



A rental company employee would need to add history to a vehicle if there is something extraordinary that happened during its most recent rental. At this portal the employee would be able to see all previous history entries and is able to add new history items. The User would be able to add text, then select the category of the update, and submit it.

3 distinct interactions

Emily Reynolds Traceability Matrix

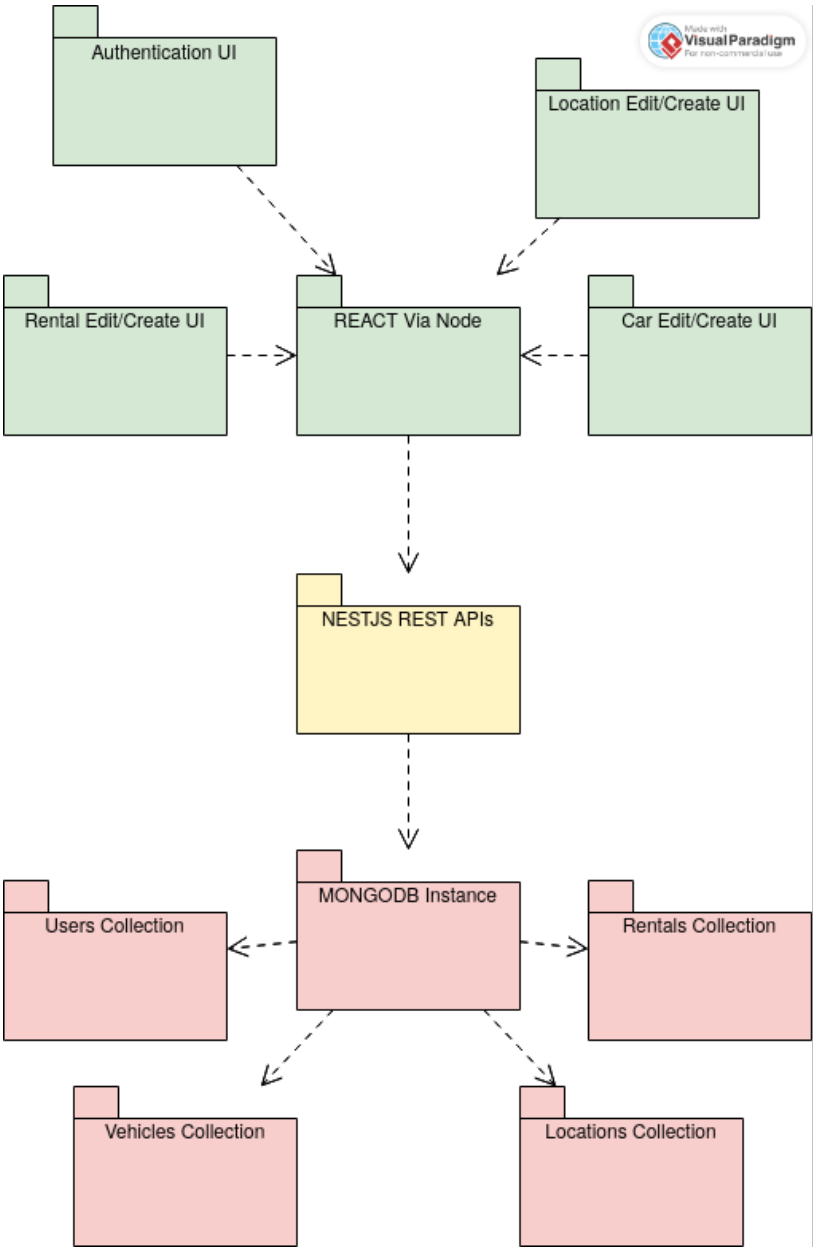
Req No	Pr io rit y	Description
REQ-1	5	Users view and select vehicle inventory at a specific pickup location.
REQ-2	5	Customers can return to any location, but must be selected at time of rental
REQ-3	3	Vehicles, Locations, and Users are manageable by admins
REQ-4	3	Rental location employees should be able to modify reservations
REQ-5	5	Service is available via webapp
REQ-6	5	Customer can select specific vehicle three to five days before rental. Same-day rentals are not guaranteed.
REQ-7	4	Rental company employees must be able to manage location of vehicles and details of customer rentals via an admin-panel

REQ-8	3	Cost is determined by pickup location, dropoff location, vehicle, and duration of rental
REQ-9	3	Payment must be made using credit or debit
NF-REQ-1	4	Should be able to be hosted on minimal server hardware - PERFORMANCE
NF-REQ-2	3	Should have a 99.9% uptime - RELIABILITY
NF-REQ-3	3	Should have modern webapp appearance that is simple to navigate for all levels of users - SUPPORTABILITY
NF-REQ-4	3	Should not visibly lag upon loading new vehicles with reasonable internet connection - PERFORMANCE
NF-REQ-5	2	Should be easily configured and support different business models in different locations and for different clients - USABILITY
UI-REQ-1	5	Must have legible design with clearly defined features
UI-REQ-2	3	Should be attractive, at least not off-putting to look at
UI-REQ-3	3	UI should utilize a simple one-column design with a navbar at the top
UI-REQ-4	3	There should be pictures of the specific car that is being offered rather than stock images
UI-REQ-5	1	User should be able to rate vehicles and view previous rentals

UC-1	Add/Remove Vehicle
UC-2	Add/Remove Employees
UC-3	Add/Remove Locations
UC-4	Modify Pricing
UC-5	View Account
UC-6	View Vehicle History
UC-7	Make Generic Reservation at a specific locations
UC-8	Make specific reservation before trip
UC-9	Pay for vehicle using credit or debit
UC-10	Pay for vehicle using third-party payment
UC-11	Accept receipt of returned vehicles
UC-12	Modify history of vehicles
UC-13	Distribute keys for rentals
UC-14	View real-time location of vehicle
UC-15	Modify reservations
UC-16	Accept customer payments via credit card
UC-17	Provide catalogs of vehicle by location

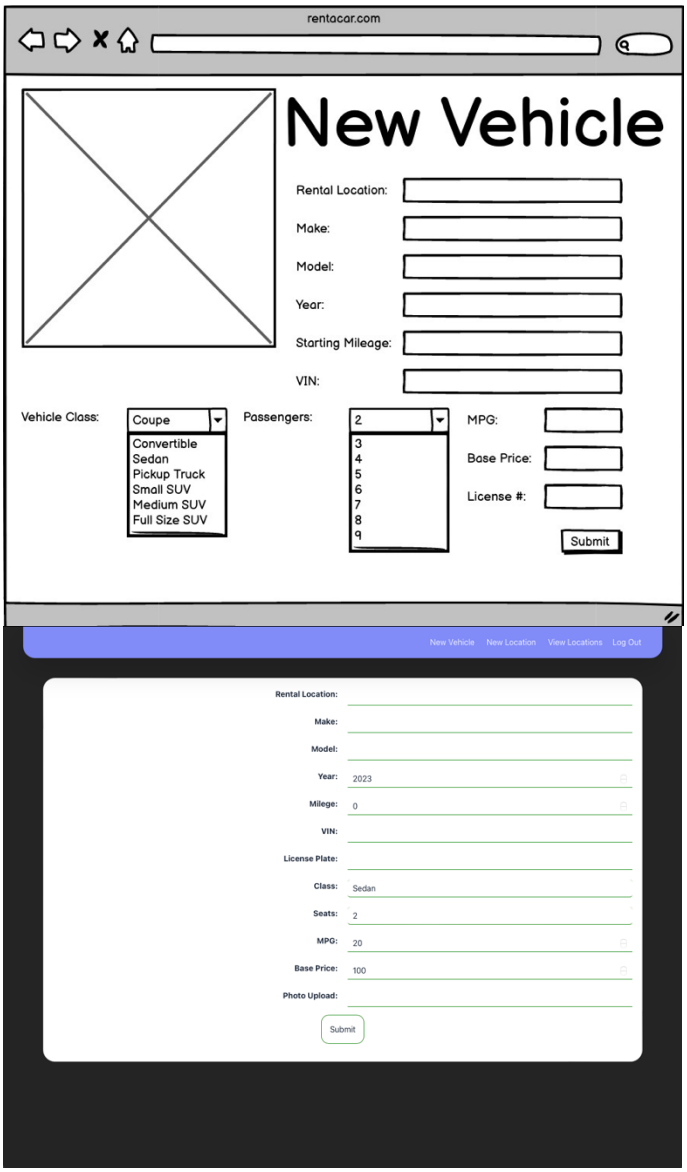
UC-18	authentication

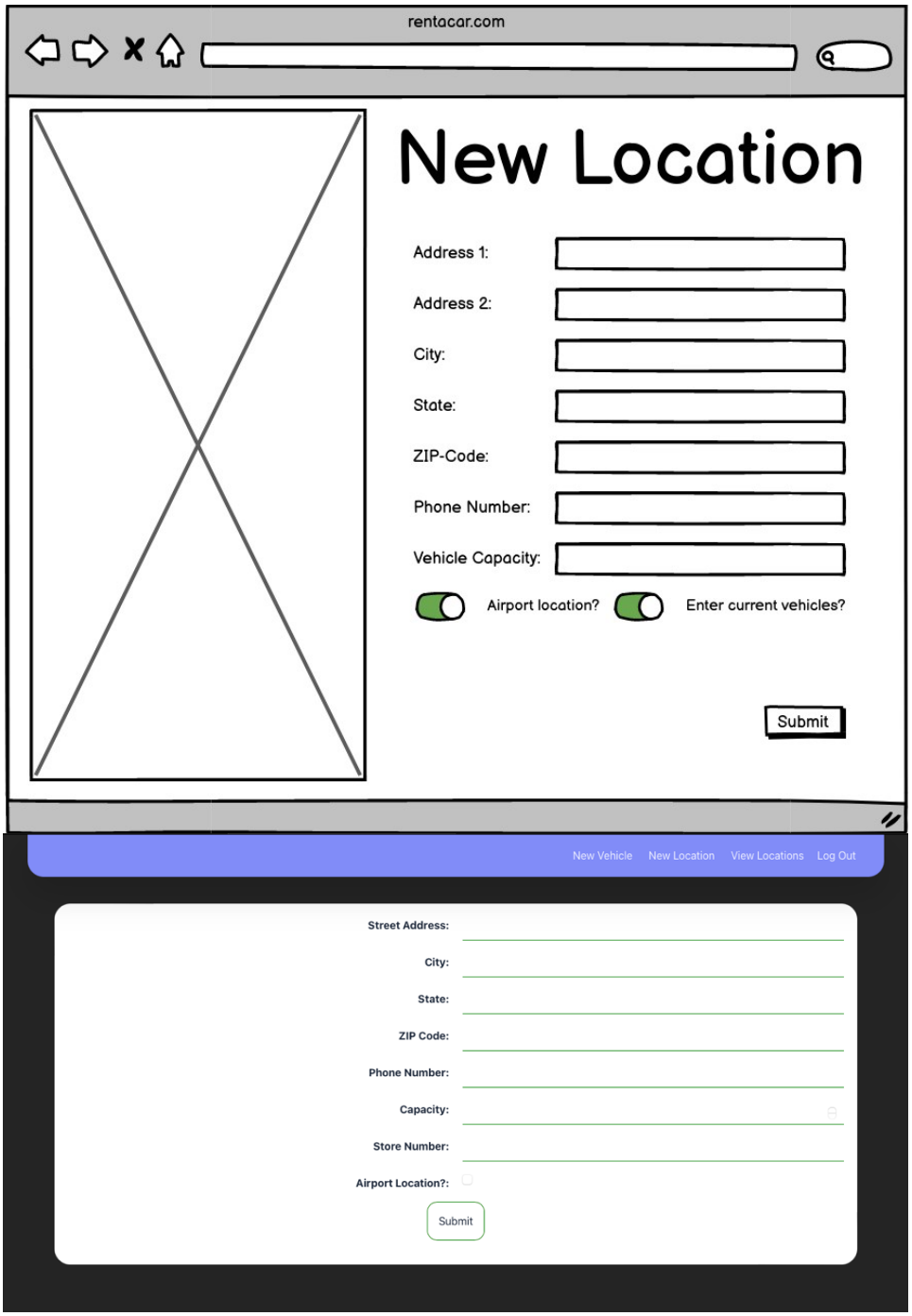
REQ#	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17	UC-18
REQ-1	5								X			X						X	
REQ-2	5											X		X					
REQ-3	5	X	X	X	X	X													
REQ-4	5															X	X		
REQ-5	5																		X
REQ-6	5							X	X	X	X								
REQ-7	4						X					X	X	X	X				
REQ-8	3				X					X	X								
REQ-9	3									X	X								
Max PW		5	5	5	5	5	4	5	5	5	5	5	4	5	4	5	5	5	5
Total PW		5	5	5	8	5	4	5	10	11	11	14	4	9	4	5	5	5	5



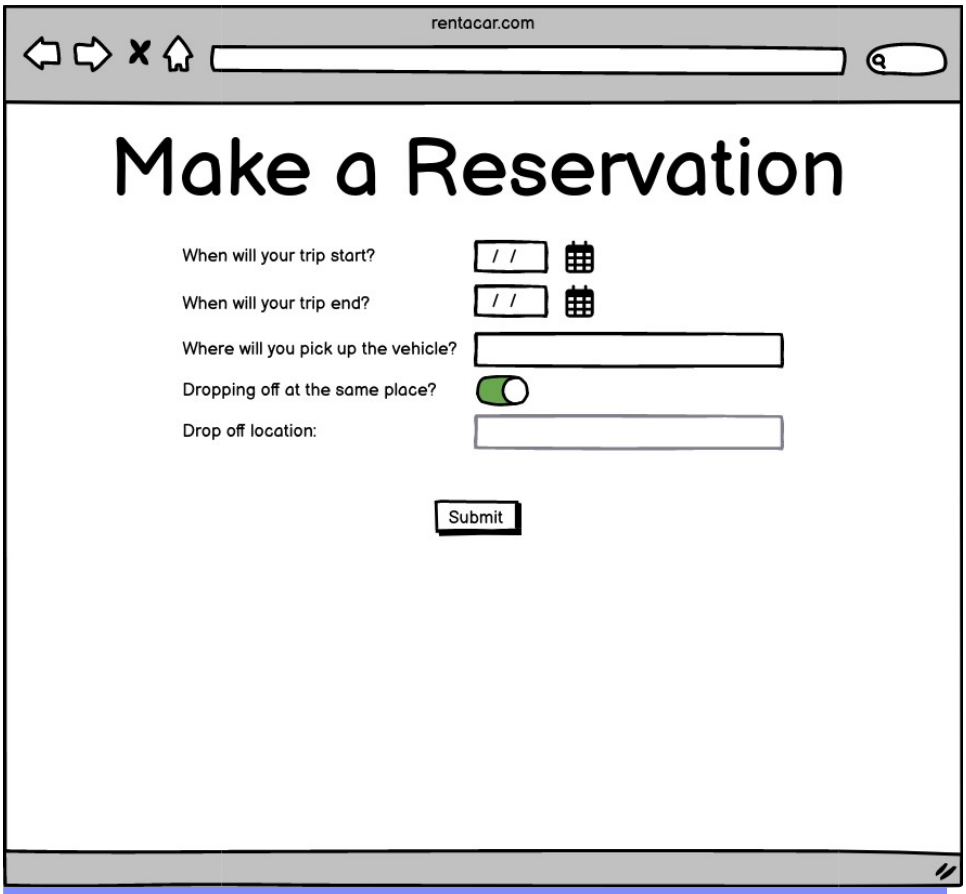
UI UPDATES

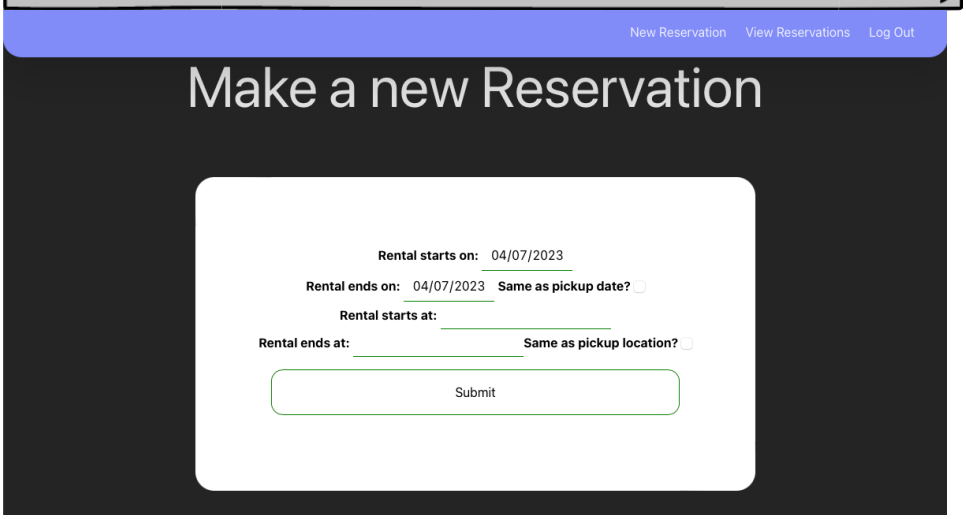
The New Vehicle screen is very close to the original idea. All fields are still present. I just changed the image field to a textbox, because storing images on the DB is not a great idea.


rentacar.com
New Vehicle
Rental Location:
Make:
Model:
Year:
Starting Mileage:
VIN:
Vehicle Class:
Coupe
Convertible
Sedan
Pickup Truck
Small SUV
Medium SUV
Full Size SUV
Passengers:
2
3
4
5
6
7
8
9
MPG:
Base Price:
License #:
Submit
New Vehicle
New Location
View Locations
Log Out
Rental Location:
Make:
Model:
Year:
2023
Milege:
0
VIN:
License Plate:
Class:
Sedan
Seats:
2
MPG:
20
Base Price:
100
Photo Upload:
Submit

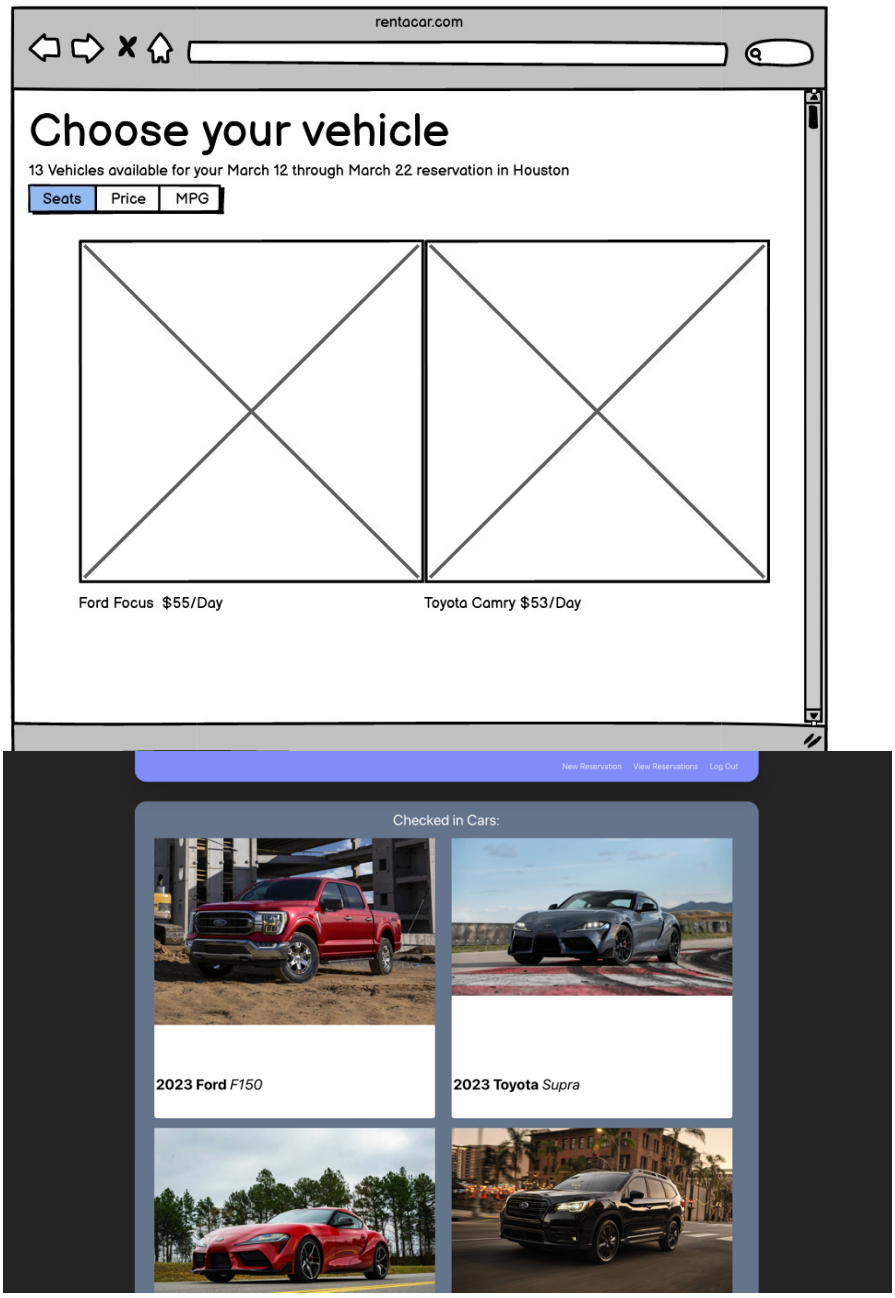


The New location page is very similar to the new vehicle page. The same things had to be changed regarding images.

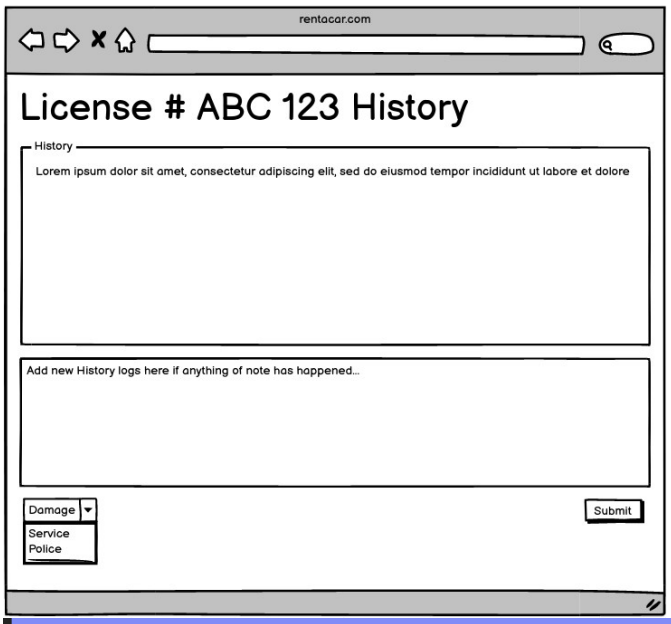


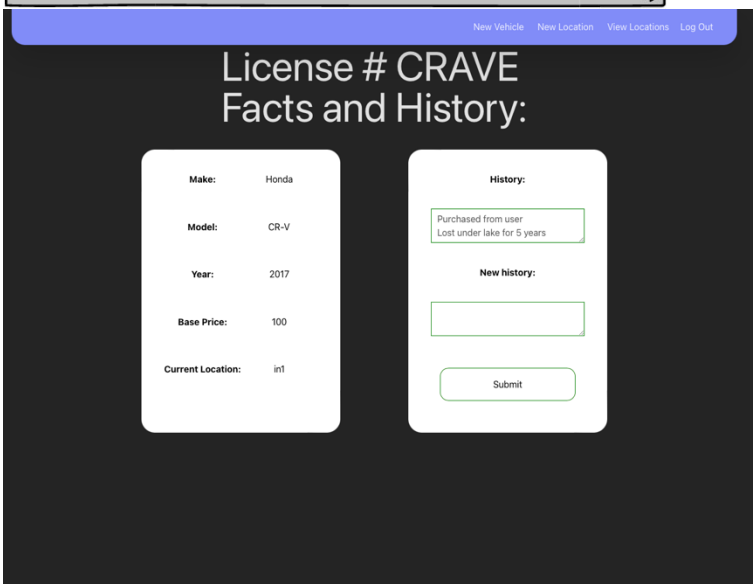


The new Reserveation page is very much the same as the initial concept. All the same fields are present. That being said, there are fields for copying both the date and location because maybe there is a one day rental within a single city.



The vehicle selection screen retains much of the same functionality as the wireframe. It has not gotten the filters yet, prices are also hidden within the click through on the vehicle in order to hopefully retain user attention longer.





The History update page now shows the vehicle's specs as well as past history items. History is stored as an array of strings on the DB, so types are not necessary

TESTS

Most testing for my application will be done via Puppeteer or some similar front-end manipulation tool. This is because this application does not implement any significantly complex algorithms.

- Test login
  - Should be able to log in to front end as

- Admin
- Employee

- Customer

- Each type will be verified by checking the contents of the navbar

- Test Create and delete items
  - Each type of collection
    - Vehicle
    - User
    - Location
    - Reservation
  - Should get a success message or redirect upon completion
  - This will test the integration with backend via front end
- Test Lifecycle of rental
  - Steps involved
    - New rental (same day)
    - View reservation
    - Add specific vehicle
    - Checkout (no payment)
    - Check back in
    - Update history (possibly)
- Test Authentication
  - If not logged in should redirect to home page
  - If logged in with wrong permissions, should redirect to valid page

Coverage of tests is nearly complete. Being able to run all of these processes should test the front end design, the back end connecting to the front end and modifying data as necessary, and the integration between the two.

In terms of testing nonfunctional requirements and UI requirements, a good strategy would be user testing with friends and family to critique usability as well as functionality.

Plan of work:

- w1 - 2: Design front end in React and connect to mongodb database using node ----
  - I'm still working on the front-end design for customers. It is coming along. The

    database connection has been tested.

- w3 - 5: build management system for rental companies

  o Create vehicles
  o Manage vehicles on location o Request maintenance

  - I'm also currently working on designing how the data will be modeled in this system

- w6 - 7: Build out the authentication system for both user types
- w8: test the main features accomplished, record the demo for the mid-term
- w9: implement employee functionality
- w10: implement customer functionality
- w11: beautify everything
- w12 - 14: writing test cases for the implemented features or continue building possible

  features you'd like to implement

- w15: record the demo for the final presentation