SmartCars System Design Document

Date: 12/16/2015
Author: Gerald Trotman
Reviewer(s): Jason Reed, Takayuki Lida

Introduction

This document defines the design for the SmartCars System which is an entire software system designed to support

This document is organized into the following sections:

1. **Overview:** summarizes the problem that is being solved and why
2. **Requirements:** enumerates the various requirements for the Asteroid Exploration System as a whole, and the sub-system requirements
3. **Module Component Diagram:** illustrates the various sub-components of the system and how their interdependencies
4. **Use Cases:** enumerates high-level use cases that the SmartCars System must support
5. **Class Diagrams:** graphs of the primary classes in each sub-system and their relationships
6. **Activity Diagrams:** shows an example of the activities involved in provisioning a new Rider Requst.
7. **Class Dictionaries:** a catalog of the classes that collectively comprise the SmartCars System and the various methods, properties, and attributes that define each class within each sub-system
8. **Sequence Diagrams:** shows examples of how users interact with the SmartCars System, and how sub-systems interacts with other sub-systems
9. **User Interface Wireframes:** these wireframes show at a high level what the MobileApplication should look like for several key screens.
10. **Implementation Details:** special considerations and explanatory observations about how the classes and sub-systems relate to each other and work
11. **Testing:** explanation of how the classes in the SmartCars System may be tested, and explain how an example testing harness would work
12. **Risks:** identifies and enumerates any potential pitfalls or deficiencies in the current implementation and issues that might arise from this implementation

Overview

Overview of the problem to be solved. What is the problem and why is it being solved? How will the resulting solution provide business value?

The goal of the SmartCars System is to create an ecosystem that can provide a next generation Uber system, where autonomous and intelligent cars provide the key component.

Because the vehicles in this system are driverless, this poses as one of the more primary problems being solved. The vehicles not only control the driving but also automatically offer their services in response to ride requests, providing limitless business value.

The system components of the SmartCars System are broken out into three subsystems:

- Mobile App
- Smart Vehicle System
- Smart Car Services

The Mobile App provides a mobile interface for customers to implement the basic functionality for the User such as create a profile an requesting a ride.

The Smart Vehicle System provides services specific to the vehicle. The system is hosted in the cloud and provides the intelligence for the car.

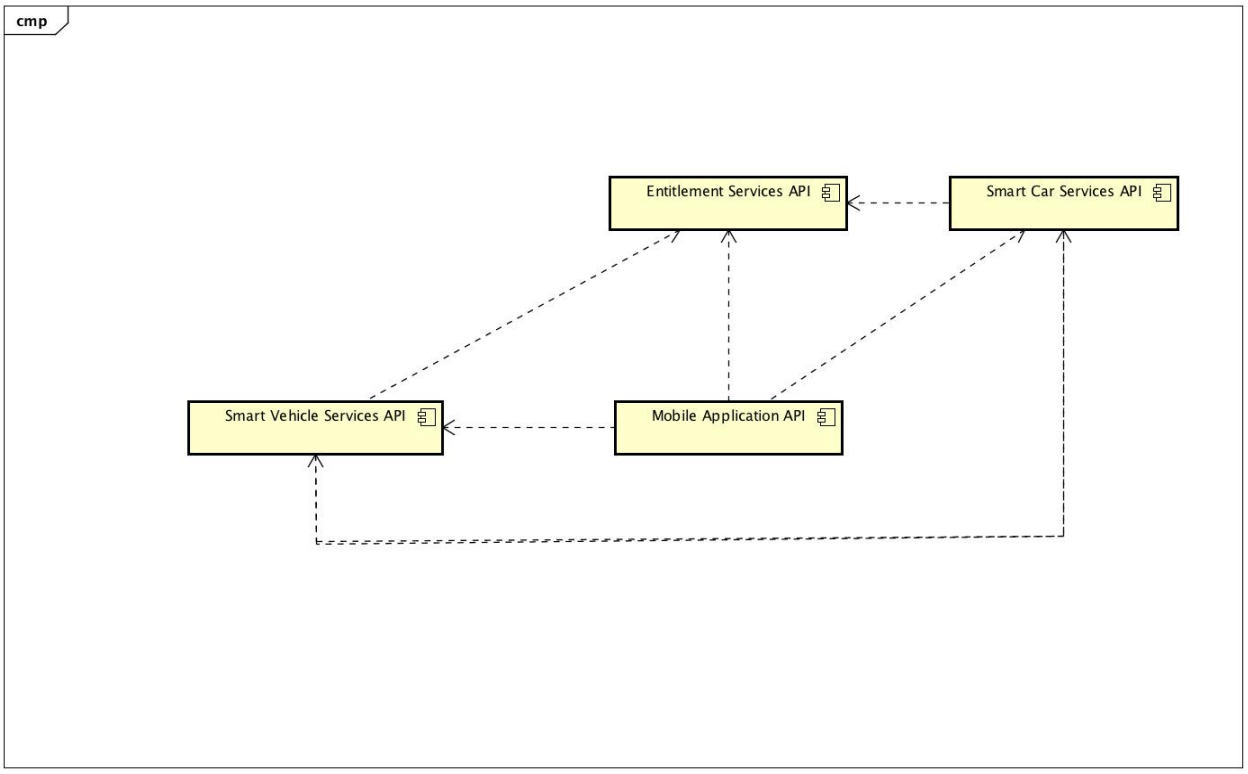
The Smart Car Services provides central functionality for the management of the Smart Car Ecosystem.

Requirements

The Requirements on this particular assignment were a lot more open than in previous assignments. But that I believe was done intentionally. However loosely defined, each service module should define a service that can be called from the other modules where appropriate (such as the MobileApp to the SmartCarServices and Smart Vehicle System, etc.). Secure the SmartCarServices using the Entitlement from Assignment 4 but don't include it in detail in the System. For the UI, we need to include a wireframe of what each of the user interfaces should look like. As far as the data itself, it requires that the data persist to a database.

Module Component Diagram

This diagram shows how the individual sub-system components within the application interact with each other.

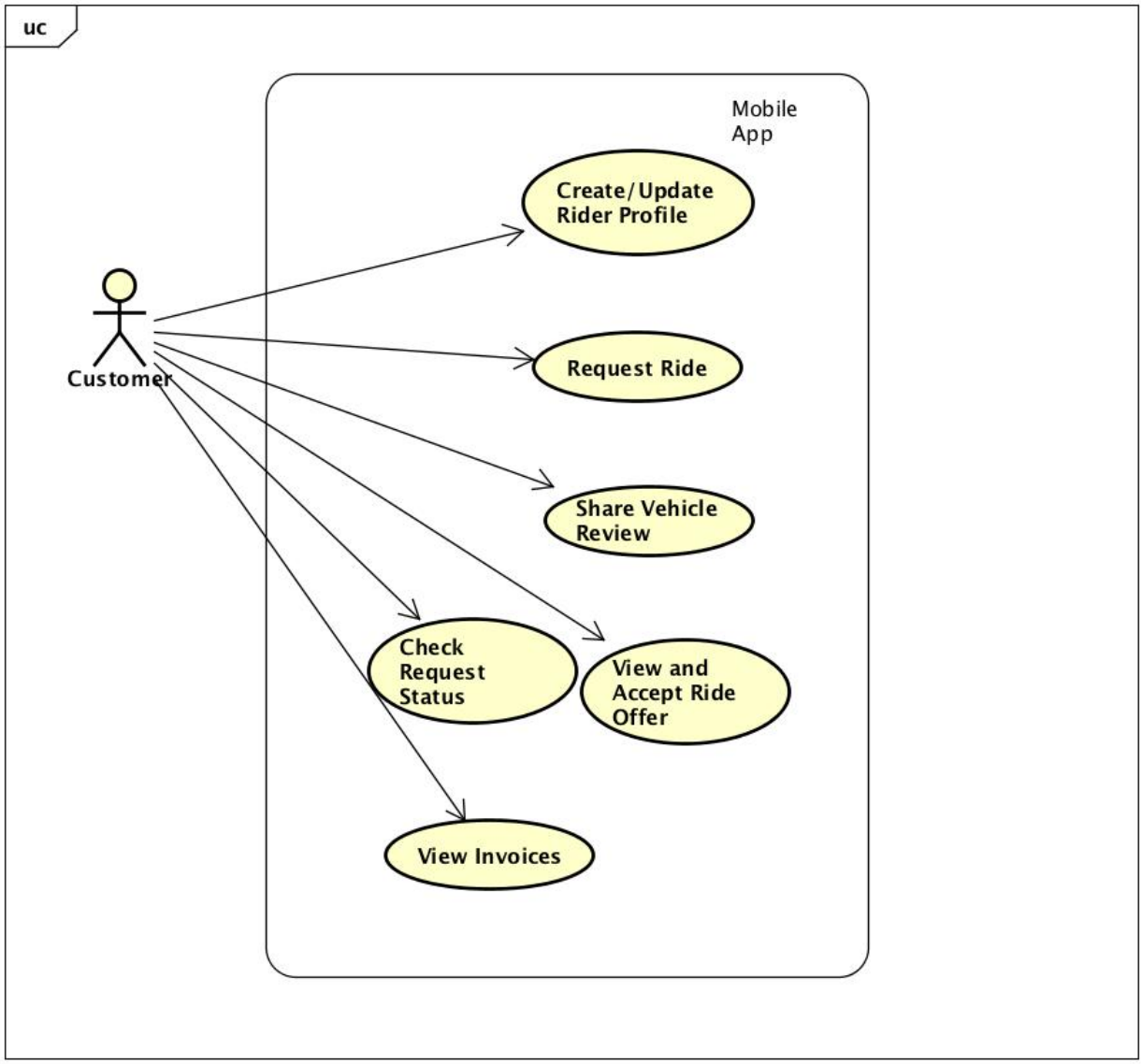


As shown, the Mobile Application API has dependencies on all other modules which shouldn't be too surprising seeing as how it is our user-interface front end for the entire application. The Smart Vehicle Services API has to communicate the various commands to the SmartCar Services. And conversely, the SmartCar Services has to make all of the system data available to the Smart Vehicle Services for just about everything. One thing to note, and that is there are no circular dependencies.

Use Cases

The following use case diagram shows the major functional use cases that each type of user of the Entitlement Service can perform. Because there are many use cases, they will be displayed in separate diagrams by sub-system.

Mobile App



Create/Update Rider Profile: The new perspective rider creates a profile that includes a user name, password, avatar and TouchID. This is the basis for how a user

is identified by the SmartCarService and Smart Vehicle System.

Request Ride: The main function of the end user. This feature is the basis for how the Rider interacts with the system. The user provides his current location and based on his profile preferences, a vehicle accepts or denies the request

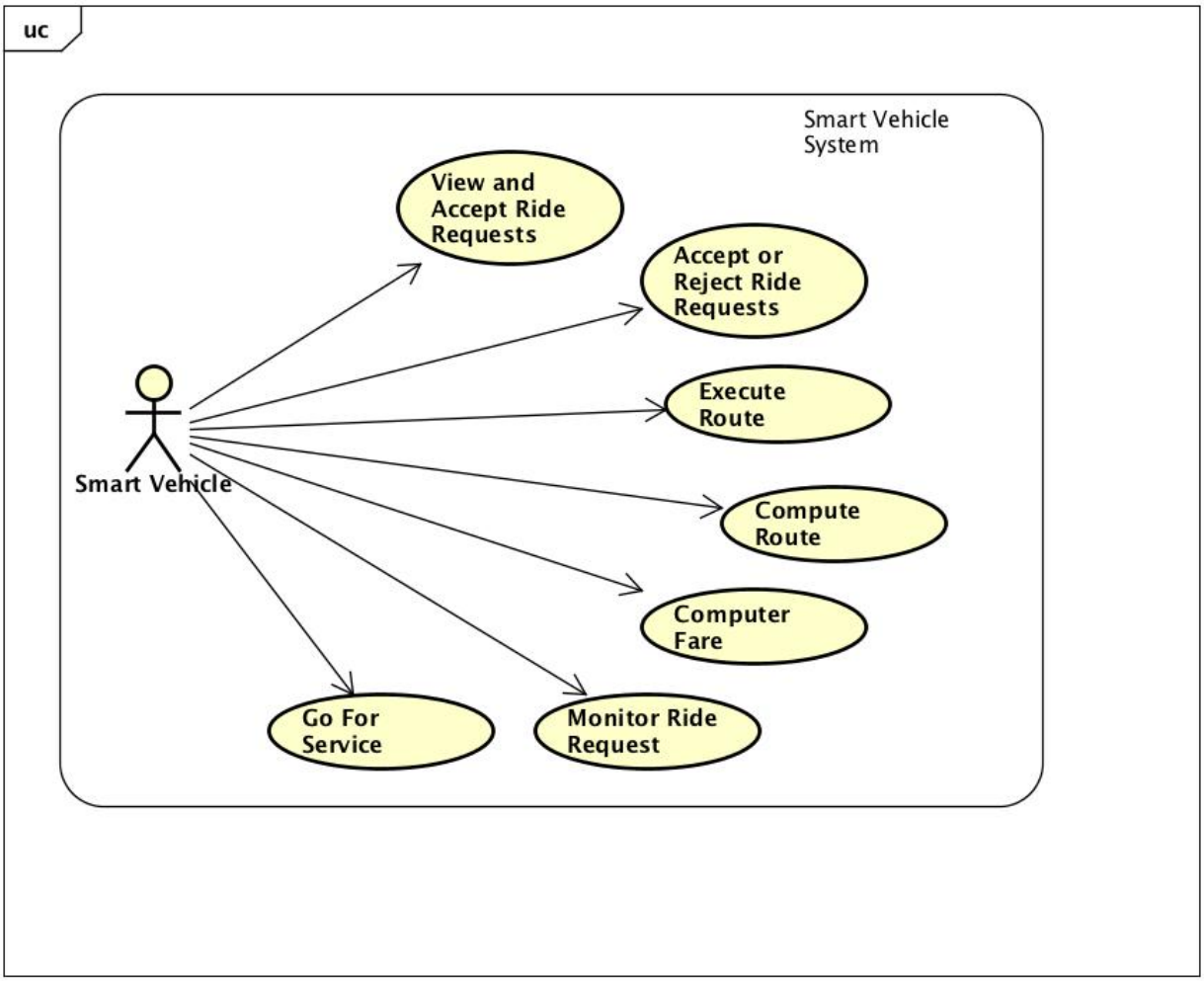
Check Request Status: This allows the user to view the status of vehicles and assess their potential to be picked up by the vehicle that meets the criteria.

View and Accept Ride Offer: The Mobile app interface generates UI objects that prompt an authenticated Rider Profile to view and/or accept ride offers.

Share Vehicle Review: As part of the completion of a Ride, A Rider has the option of commenting on the experience contributing back to the ecosystem of the SmartCars Service and the over all performance quality.

Review Invoices: After the Rider has reached his destination, the Rider can view the cost of service for his or her records along with things like date, the vehicle id.

Smart Vehicle System



Monitor Ride Request: This allows the smart vehicle the ability to assess Users based on their Rider Profile if they meet their criteria or not as they de-queue the requests.

View and Accept Ride Request: Upon verifying the criteria of the Rider Profile and calculating the distance to the potential Rider location, it makes an offer that gives itself the highest likelihood of being accepted.

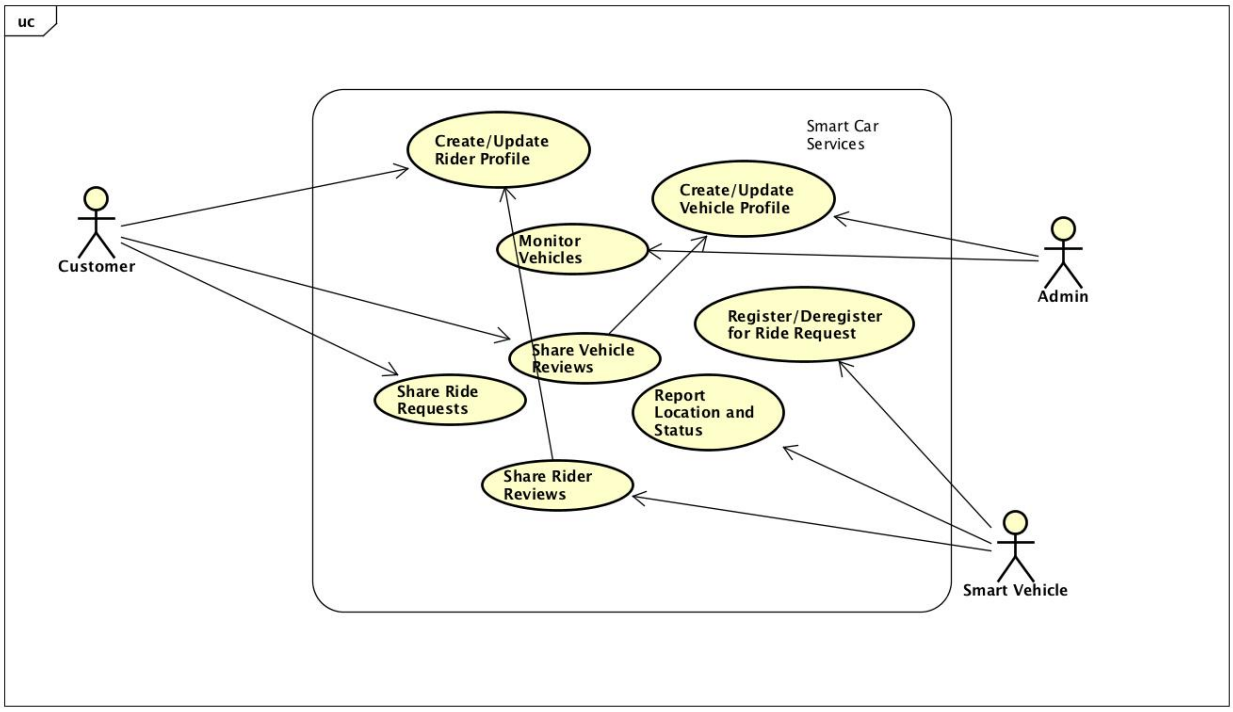
Compute Route: The Smart Vehicle computes the distance in miles from the potential Rider's current location to its requested destination after assessing the shortest route.

Compute Fare: The Smart Vehicle computes the price/miles from the potential Rider's current location to the required destination after assessing the shortes route.

Go For Service: The Smart Vehicle uses an internal algorithm that assesses the safety level and/or its fuel level. If these levels go below the threshold, the Vehicle

Execute Route: This is the actually implementation of the vehicle picking up a Rider Request from its current location to its target destination.

Smart Car Services



Create/Update Rider Profile: In it contains information about registered riders including their reviews and ratings, their preferences of vehicle, the most they are willing to spend for a ride, and the urgency.

Monitor Vehicles: Track and provide a means of knowing where riders and vehicles are and their status.

Create/Update Vehicle Profile: The SmartCar Service maintains an inventory of vehicles that maintain

Share Ride Requests: The SmartCar Service implements a method for distributing the request for service to other vehicles to increase the awareness that a Rider is requesting a ride increasing the chances of the Rider Request wait time to be low.

Share Vehicle Reviews: This allows the SmartCars Service to store customer comments and ratings and distribute them amongst other authorized Rider Profiles which adds scrutiny to which Vehicle gets approved or not to be picked up.

Share Rider Reviews: This allows the Smart Vehicle System to store comments and ratings about Riders and distribute them amongst other Smart Vehicles which adds scrutiny to which Rider gets selected or not.

Report Location Status: This allows the SmartCar Service determine things like Route and Fare upon obtaining the status of the Rider in relation to the Smart Vehicle.

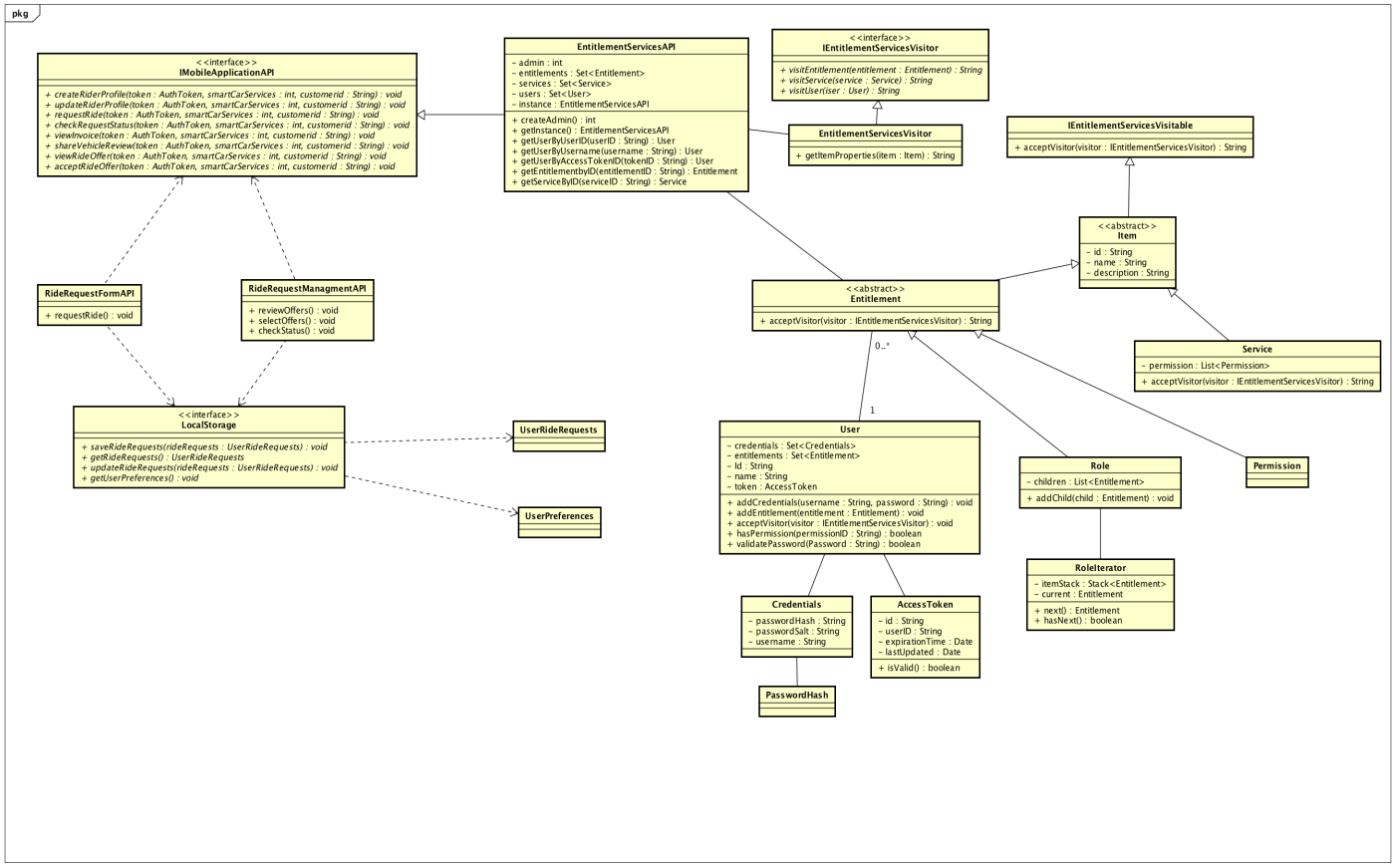
Implementation

Class Diagram

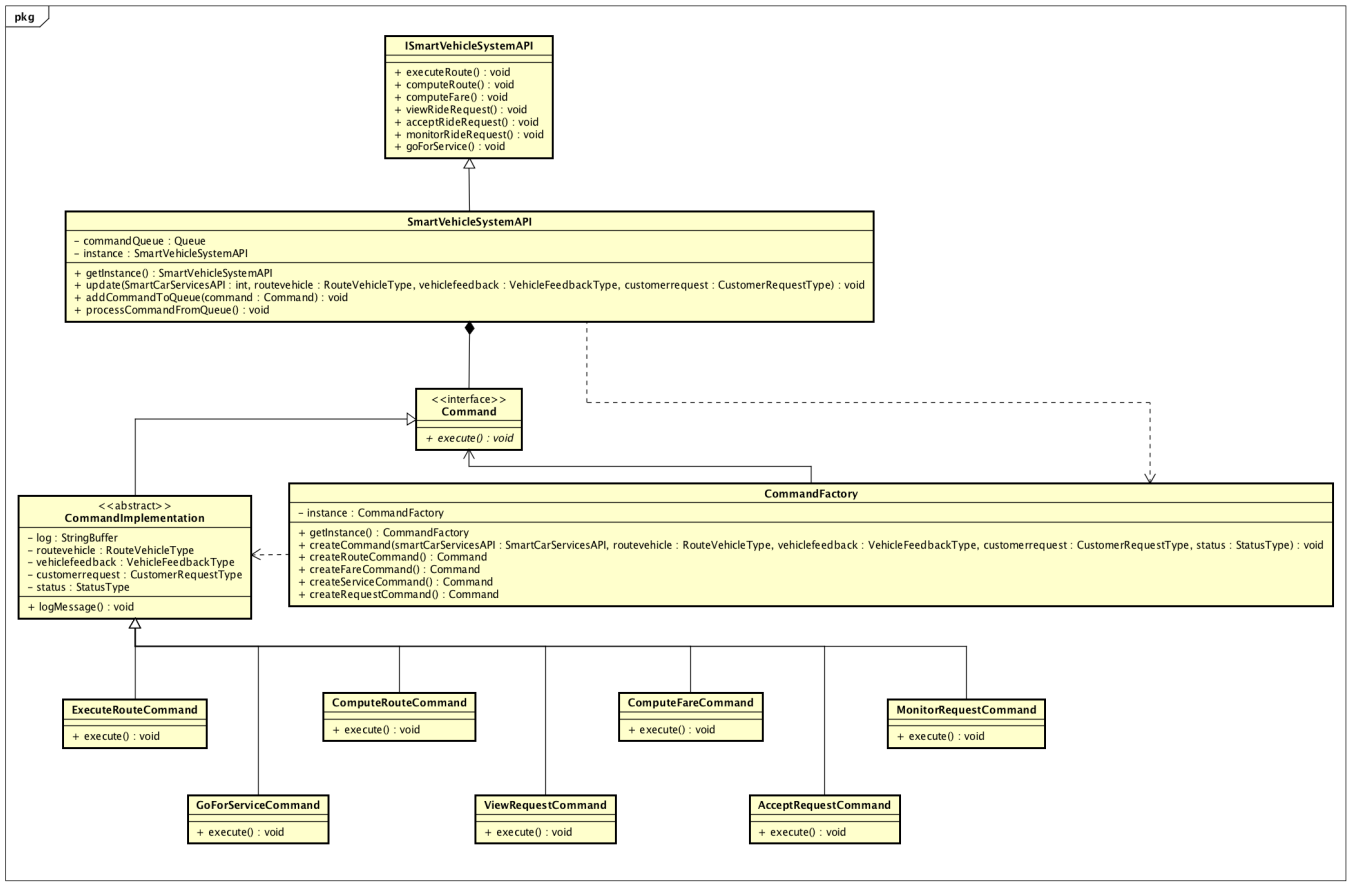
Each subsystem will have its own class diagram. Note…The expectation is that these systems will use best practices in developing and using this concept of the Model-View-Controller design pattern and its supporting framework and database.

The details of each class, interface, and enumeration in each sub-systems's class diagram will be explained in the Class Dictionary in the next section.

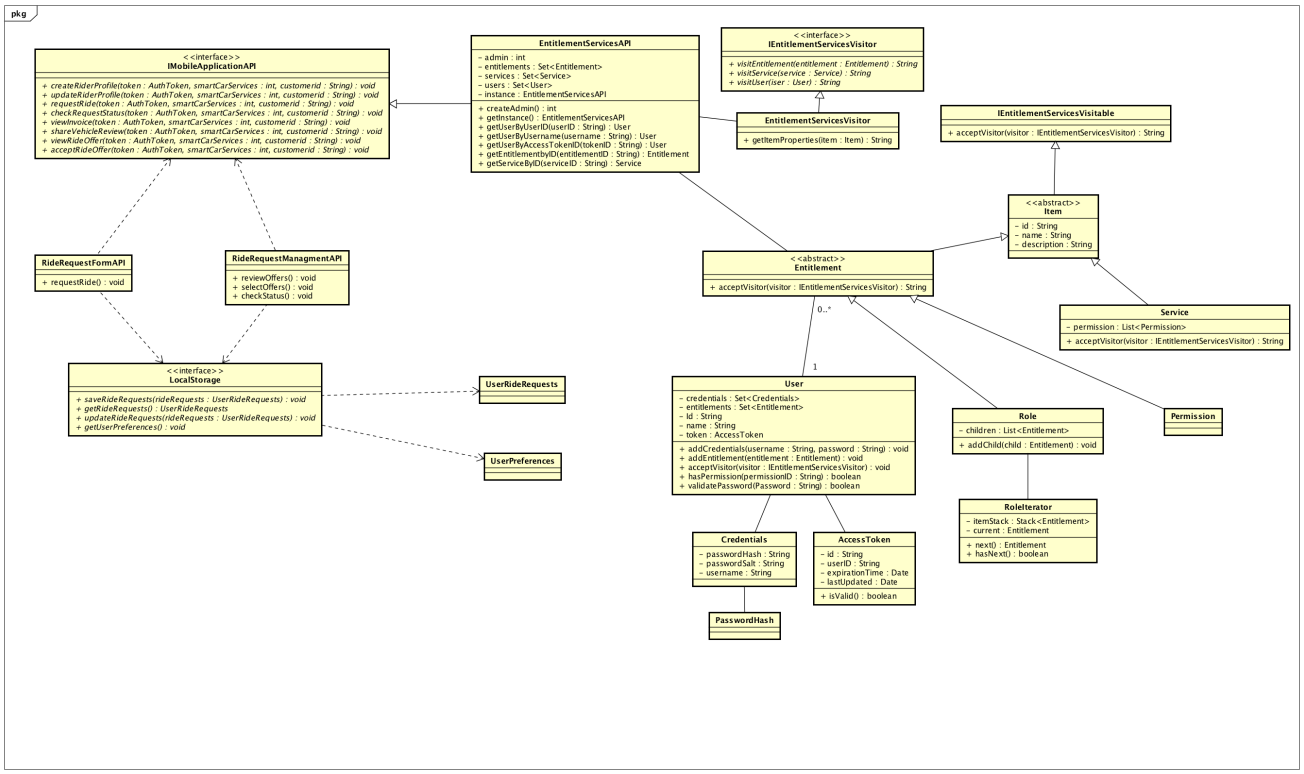
SmartCars System Design Document
CSCI E-97 Assignment 5
Gerald Trotman



SmartCars System Design Document
CSCI E-97 Assignment 5
Gerald Trotman

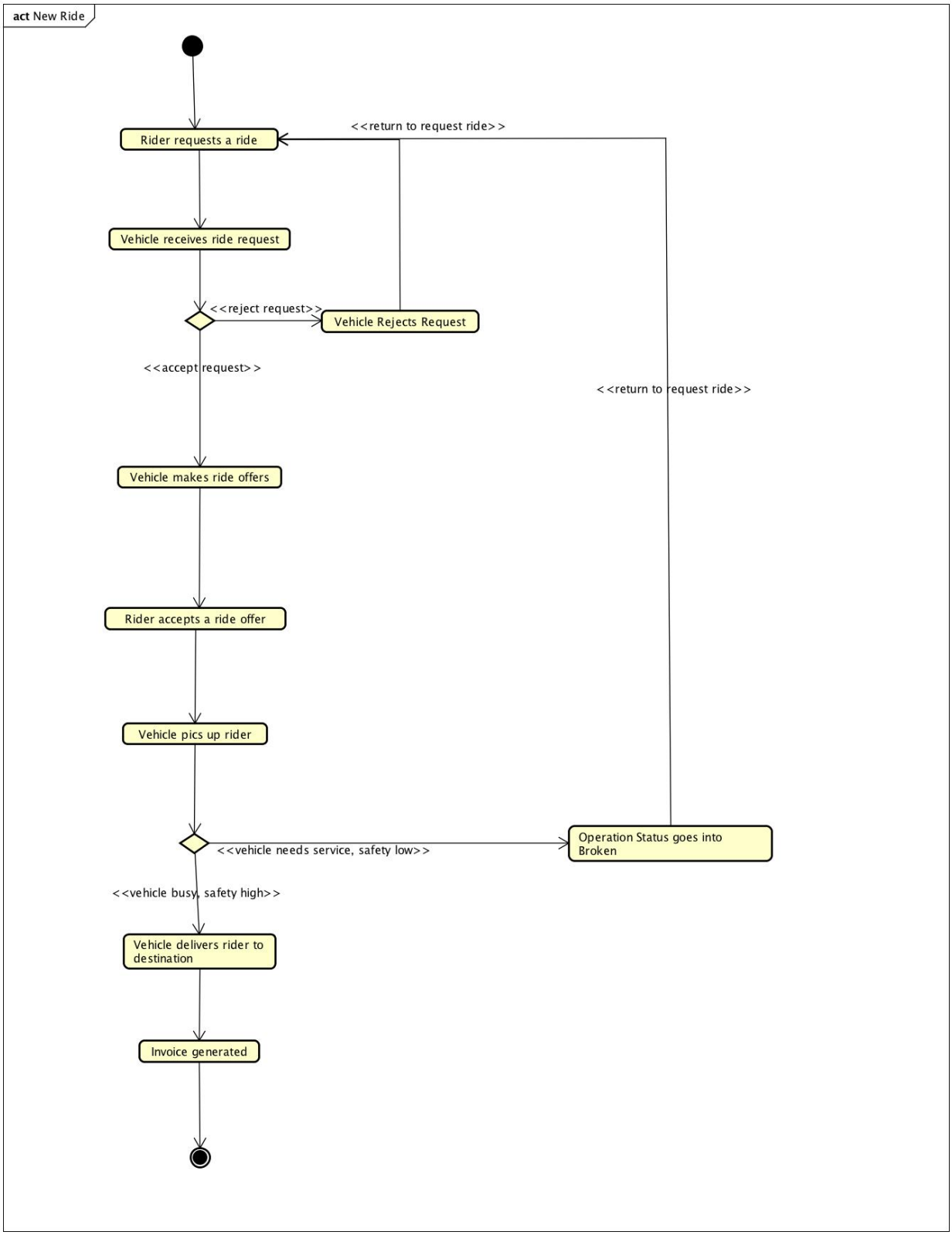


SmartCars System Design Document
CSCI E-97 Assignment 5
Gerald Trotman



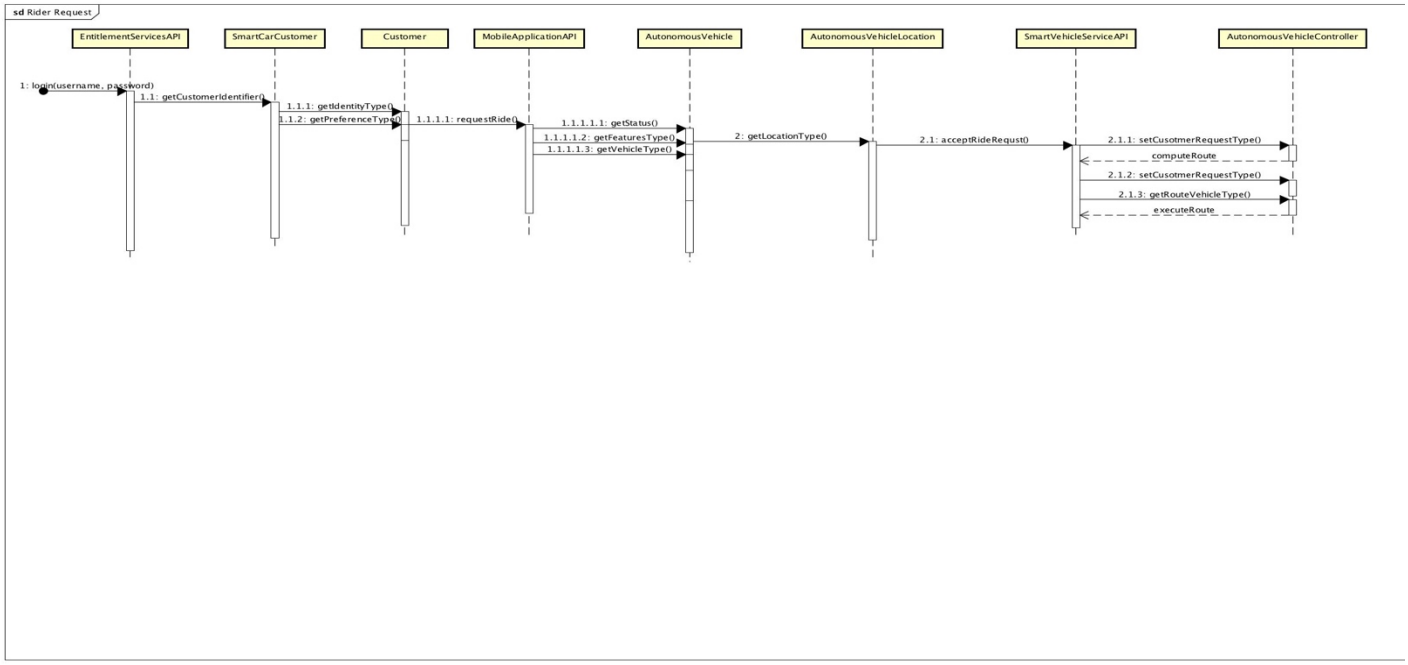
Activity Diagram

The following example activity diagram shows the logical process to provision a new ride request by a user of the system.

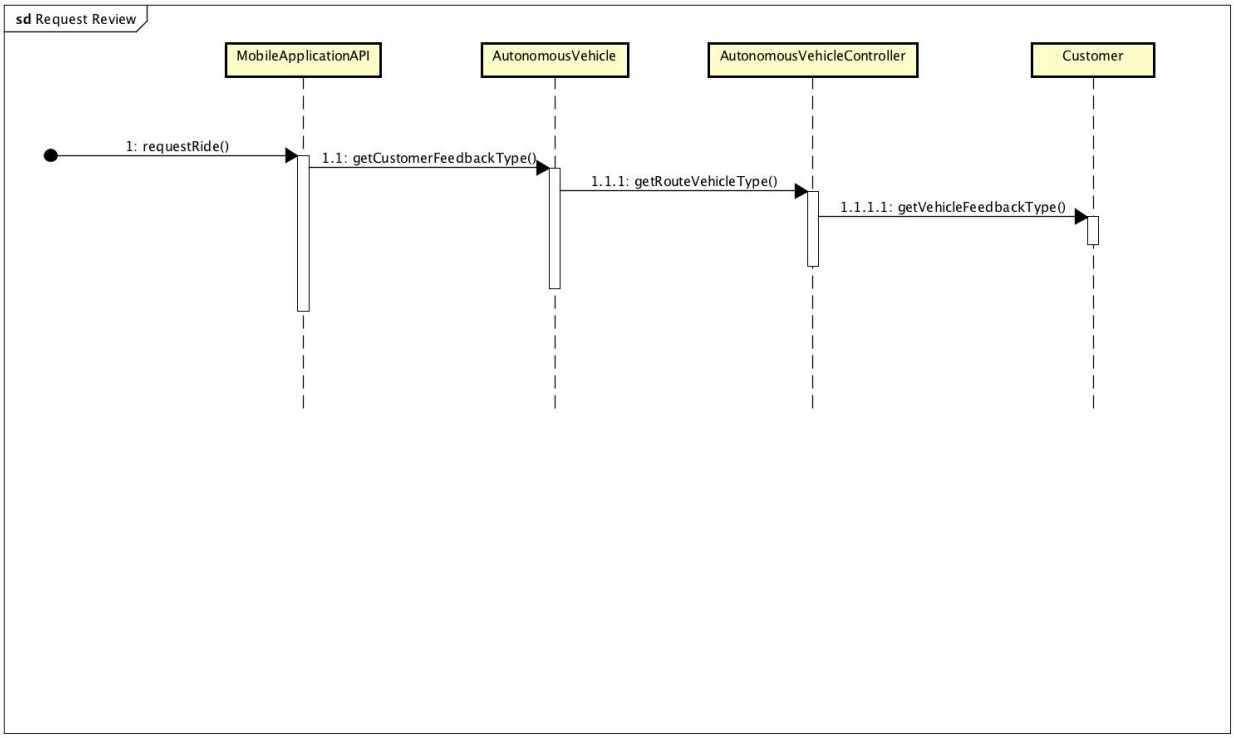


Sequence Diagrams

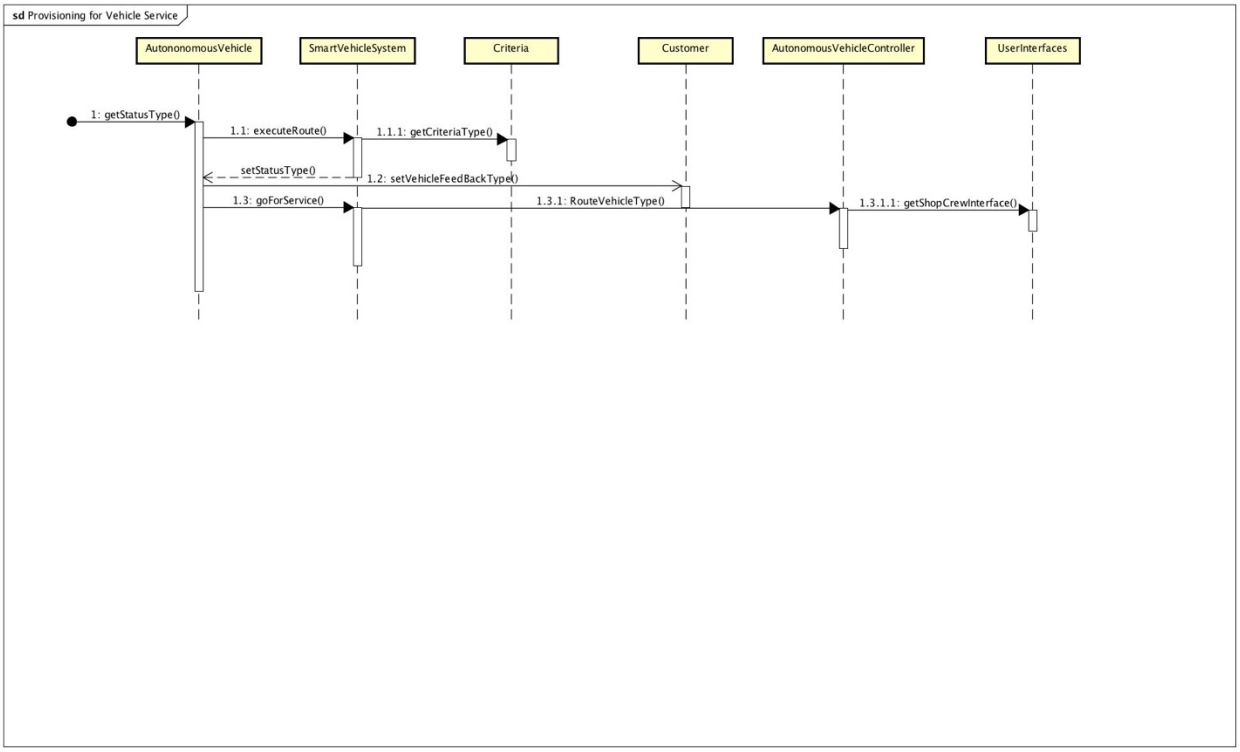
This sequence diagram represents the message flow detailing how riders can send ride requests, receive ride offers, and accept a ride offer:



This shows the message flow for how rider and vehicle reviews are processed:



This shows the provision of a vehicle. I thought it would be interesting to see what happens when a new vehicle failed so I chose to sequence that:



Class Dictionary

This section specifies the class dictionaries for each sub-system in the SmartCars System. In attempts of being somewhat brief, I will try to omit simple accessor/mutator methods as well as overridden methods in classes implementing interfaces or abstract classes.

Package: cscie97.asn5.service

ISmartCarServicesAPI <<interface>>

This interface defines the primary service contract for any classes that intend to implement these methods and act as the concrete implementation of the SmartCarServicesAPI. All methods require an authenticated access via Entitlement Service API; methids must pass an AuthToken, which is checked before continuing. Allows authenticated users access to manage

all aspects of SmartCar. This interface and its implementing class, collectively wok as a "___ pattern" – only the pubic methods here are accessible outside the package.

Methods

Method Name	**Signature**	**Description**
createRiderProfile	(AuthToken:token, customer: Customer):void	Clients may create new customer objects independently, but this method must be called to add the Customer to the inventory of profiles(also persists the newly created Customer to the database)
monitorVehicles	(AuthToken:token):List <AutonomousVehicle>	Returns all vehicles one has authentication to monitor
updateRiderProfile	(AuthToken:token, String, customerid, Customer: updateRiderProfile)void	Finds the Customer in the inventory of profiles with the specific ID and overwrites that objects properties with those of the updateRiderProfile. Should also persist the changes to the database.
createVehicleProfile	(AuthToken:token, AutonomousVehicle:vehicle):void	Admin may create new Vehicle objects independently but this method must be called to add the Vehicle to the inventory (also persists the newly created Vehicle to the database).
updateVehicleProfile	(AuthToken:token, String: autonomousvehicleid, AutonmousVehicle:updateVehicleProfile):void	Finds the Vehicle in the inventory of vehicles with the specific ID and overwrites that objects properties with those of the updateVehicleProfile. Should also persist the changes to the database.
shareRiderRequests	(AuthToken:token, VehicleRequests: shareRideRequests): list<VehicleRequests>	Returns customer's ride request and enables authentication for other registered users to view customer's ride requests based on location.
shareVehicleReviews	(AuthToken:token, AutonomousVehicle:shareVehicleReviews):list<Customer>	Returns customer reviews of the vehicle for authentication enabled users.
shareRiderReviews	(AuthToken:token, Customer:shareRideReviews):list<AutonomousVehcile>	Returns customer reviews of ride of vehicle for authentication enabled users.
reportLocationStatus	(AccessToken:token, AutonomousVehicle:	Returns all the enumeration values of status of Location of the customer's ride.

	autonimousvehicleid, LocationType: location) : Set<AutonomousVehicleLocation>	
registerObserver	(ISmartCarService:observer):void	This method updates the SmartCarServices whenever a change has taken place

ISmartCarServicesObserver <<interface>>

ISmartCarServicesObsever is an interface that defines a method to notify registered observers of the status updates in Customers, VehicleRequests, and the AutonomousVehicle itself. These classes implements this interface which means that all are able to send status update notifications to the observers. SmartVehicleServices is the only Observer of the notifications.

Methods

Method Name	**Signature**	**Description**
update	(RouteVehicleType: routevehicle, VehicleFeedbackType:vehiclefeedback, CustomerRequestType: customerrequest): void	

SmartCarServicesAPI, implements ISmartCarServicesAPI

This concrete implementation of the ISmartCarServicesAPI follows the Singleton design pattern; to obtain an instance to use, the getInstance() method will return the sole instance of the object.

Properties

Property Name	**Signature**	**Description**
instance<<private>>	():SmartCarServicesAPI	Singleton Instance of the API; only one exists at a time.
vehicle	List<AutonomousVehicles>	Contains the creations of all the Vehicles in the inventory.
customer	List<Customers>	Contains the creation of Users of the SmartCar Service.

userinterface	List<UserInterfaces>	Contains all the UserInterfaces in the inventory.
invoice	List<Invoices>	Contains all the Invoices in the inventory.
criteria	List<Criteria>	Contains all the Criteria in the inventory.
vehiclerequests	List<VehicleRequests>	Contains all the VehicleRequests in the inventory.

Methods

Method Name	**Signature**	**Description**
getInstance<<static, synchronized>>	(): ISmartCarServicesAPI	Used to obtain a Single instance of the API for client use.
notifyObserver	(RouteVehicleType: routevehicle, VehicleFeedbackType:vehiclefeedback, CustomerRequestType: customerrequest): void	

SmartCarCustomer <<abstract>>

This abstract class allows for giving each customer and its corresponding attributes a unique identifier and a corresponding method for getting back that customer.

Properties

Property Name	**Type**	**Description**
customerid	String	The unique identifier given to every single instance of a SmartCarCustomer object.

Methods

Method Name	**Signature**	**Description**
getCustomerIdentifier	(): String	This method returns the unique identifier that links the customer to its list of attributes

Customer

This represents the various pieces of information that go into a customer profile for our SmartCar System. This data persists and writes all the information to the database.

Properties

Property Name	Type	Description
Preference	PreferenceType	The specific preferences of the customer
identity	IdentityType	The various identities that make a customer
vehiclefeedback	VehicleFeedbackType	The various attributes that comprise vehicle feedback from the customer

Methods

Method Name	Signature	Description
getPreferenceType	():PreferenceType	Returns the String name of the enumeration properties of preferences
getIdentityType	():IdentityType	Returns the String name of the enumeration properties of identity
getVehicleFeedbackType	():VehicleFeedbackType	Returns the String name of the enumeration properties of feedback

PreferenceType<<enum>>

An enumeration representing the types of Preferences the customer can choose from in deciding how to optimize their SmartCar Service experience.

Properties

Property Name	Type	Description
MIN_SPEED	PreferenceType	The minimum speed the customer would like for their car service
MIN_EFFICIENCY	PreferenceType	The minimum efficiency the customer would like for their car service
MIN_TRIM_LEVEL	PreferenceType	The minimum trim level the customer has set as his preference
MAX_RATE	PreferenceType	The maximum rate the customer is willing to

		pay for service

IdentityType <<enum>>

An enumeration representing the different forms of identification that is comprised of a customers profile in being created.

Properties

Property Name	Type	Description
THUMB_PRINT	IdentityType	The thumbprint data is added to the customers profile used to strengthen security and uniquely identify user.
PICTURE	IdentityType	The picture data is added to the customer as an avatar which in combination with other identity types contributes to uniquely identifying the user.
NAME	IdentityType	The chosen user name that is to be displayed for the customer profile

VehicleFeedbackType <<enum>>

Properties

Property Name	Type	Description
AVERAGE	VehicleFeedbackType	Stores the customers numerical rating of the driving experience on average.
COUNT	VehicleFeedbackType	Stores the amount of transactions the smart car system has registered to the customer profile
REVIEW	VehicleFeedbackType	Stores the customers review of his riding experiences.
PROBLEM_REPORT	VehicleFeedbackType	Stores the customers reported complaints of ride experience.

VehicleRequests <<abstract>>

This class represents the various methods and attributes that both the Invoice and Criteria class share that get implemented for one to both initiate and carry out a ride from the SmartVehicle System.

Properties

Property Name	Type	Description
invoiceid	String	The unique identifier that tags each Invoice object and its corresponding enumeration data
criteriaid	String	The unique identifier that tags each Critera object and its corresponding enumeration data
location	String	The string that represents customers current location.
destination	String	The string that represents the customers target destination.
pickupdate	Date	The date object representation of when customer was picked up.
numberofriders	Int	The number of rides the smart car services stores as a record.

Methods

Method Name	Signature	Description
getInvoiceIdentifier	():String	Method that gets the appropriate Invoice enumerated object to be linked with Customer Profile.
getCriteriaIdentifier	():String	Method that gets the appropriate Criteria enumerated object to be linked with Customer Profile.
getRequestLocation	():String	Method that passes the current location attached to the Customer Profile
getRequestDestination	():String	Returns the location the Customer Profile specifies to be the ending location.
getPickUpDate	():Date	Returns the date and time the Customer Profile requested service

getNumberOfRiders	():Int	Returns the number of Riders the Customer Profile specified for service.

Invoice

In addition to the abstract attributes of the VehicleRequest Class, this Class gets the enumerated attributes that comprise an invoice that ties to a customer represented in the SmartCar Service.

Properties

Property Name	Type	Description
invoice	InvoiceType	This represents the elements that make up an invoice.

Methods

Method Name	Signature	Description
getInvoiceType	(): InvoiceType	This returns the invoice and all that its type is comprised of.

InvoiceType <<enum>>

An enumeration that represents the attributes of what make up an invoice object.

Properties

Property Name	Type	Description
VEHICLE_ID	InvoiceType	The unique identifier of the vehicle
DISTANCE	InvoiceType	The calculated distance from the start location to the destination
RATE	InvoiceType	The rate specified by the vehicle system
START_DATE	InvoiceType	The date object that gives timestamp from point of pick up of customer.
END_DATE	InvoiceType	The date object that gives timestamp for the destination specified by the customer

Criteria

In addition to the abstract attributes of the VehicleRequest Class, this Class gets the enumerated attributes that comprise an invoice that ties to a customer represented in the SmartCar Service.

Properties

Property Name	Type	Description
criteria	CriteriaType	This represents the elements that represent a Criteria for a ride requested by Customer

Methods

Method Name	Signature	Description
getCriteriaType	():CriteriaType	This returns the criteria and all that its type is comprised of.

CriteriaType <<enum>>

An enumeration that represents the attributes of what make up a Criteria object.

Properties

Property Name	Type	Description
CAPACITY	CriteriaType	The amout of space the Customer specifies while making ride request
MIN_SPEED	CriteriaType	The minimum speed the Customer specifies during requested ride
MIN_EFFICIENCY	CriteriaType	The minimum efficiency specified by the Customer
MIN_TRIM_LEVEL	CriteriaType	The type of vehicle the Customer specifies as a preference for his ride
MAX_RATE	CriteriaType	The most the Customer specifies he is willing to pay for a given ride

URGENCY	CriteriaType	How fast the Customer expects his vehicle to arrive for pick up.

Inventory <<abstract>>

This class represents and manages the system of vehicles themselves. It manages things like their appearance, their status, and monitors their route. It represents a collection of the SmartVehicles themselves and caputres their collected data to be reported back to the SmartCar Services.

Properties

Property Name	Type	Description
autonimousvehicleid	String	The unique identifier given to every single instance of an AutonimousVehicle object

Methods

Method Name	Signature	Description
getAutonomousVehicleIdentifier	():String	This method returns the unique identifier that links the autonomousvehicle to its list of attributes

AutonomousVehicle

This concrete class represents the AutonomousVehicle object. In it are the attributes and services that an autonomousvehicle object is comprised of.

Properties

Property Name	Type	Description
status	StatusType	The StatusType enum values that represent the state of the vehicle
customerfeedback	CustomerFeedbackType	The CustomerFeedbackType enum values that represent comments about the vehicle
features	FeaturesType	The FeatureTypes enum values that represent the features of the vehicle

vehicle	VehicleType	The VehicleTypes enum values that represent the type of vehicle

Methods

Method Name	Signature	Description
getStatusType	():StatusType	The method that returns the various status information of the vehicle.
getCustomerFeedbackType	():CustomerFeedbackType	The method that returns the review content for a particular vehicle.
getFeaturesType	():FeaturesType	The method that returns the specific features of a particular vehicle.
getVehicleType	():VehicleType	The method that returns the specific type of vehicle

StatusType <<enum>>

An enumeration that represents the attributes of what make up an AutonomousVehicle object's status.

Properties

Property Name	Type	Description
FUEL_LEVEL	StatusType	The amount of fuel being reported by the vehicle
SAFETY_STATUS	StatusType	The safety level of the vehicle
OPERATIONAL_STATUS	StatusType	The level of operational functionality being reported by the vehicle

CustomerFeedbackType <<enum>>

An enumeration that represents the attributes of what make up an

AutonomousVehicle object's CustomerFeedback.

Properties

Property Name	Type	Description
AVERAGE	CustomerFeedbackType	The average rating of a particular vehicle
COUNT	CustomerFeedbackType	The number of feedback given on a particular vehicle
REVIEW	CustomerFeedbackType	The content of feedback given on a particular vehicle
PROBLEM_REPORT	CustomerFeedbackType	Specific warnings or issues on a particular vehicle most helpful for maintenance of vehicle.

FeaturesType <<enum>>

An enumeration that represents the attributes of what make up an AutonomousVehicle object's features.

Properties

Property Name	Type	Description
TRIM_LEVEL	FeauresType	The trim of a particular vehicle
CAPACITY	FeaturesType	The capacity of a particular vehicle
RANGE	FeaturesType	The range of a particular vehicle
COLOR	FeaturesType	The color of a particular vehicle
MAX_SPEED	FeaturesType	The maximum speed a particular vehicle is rated for.
EFFICIENCY	FeaturesType	The efficiency of a particular vehicle.

VehicleType <<enum>>

An enumeration that represents the attributes of what make up an AutonomousVehicle object's vehicle type.

Properties

Property Name	Type	Description
CAR	VehicleType	Represents the vechile as a car
TRUCK	VehicleType	Represents the vehicle as a truck
VAN	VehicleType	Represents the vehicle as a van
BUS	VehicleType	Represents the vehicle as a bus
DRONE	VehicleType	Represents the vehicle as a drone

AutonomousVehicleLocation

This class is an associated class of the AutonomousVehicle that manages an enumeration representing the attributes of location information of the vehicle.

Properties

Property Name	Type	Description
location	LocationType	The LocationType enum value that the attributes of AutonomousVehicleLocation correspond to.

Methods

Method Name	Signature	Description
getLocationType	():LocationType	This method gets the various attributes of what the AutonomousVehicleLocation are comprised of.

LocationType <<enum>>

An enumeration that represents the attributes of what make up an AutonomousVehicle's location.

Properties

Property Name	Type	Description
LOCATION	LocationType	This keeps the record of the location of the vehicle

TIME	LocationType	This keeps record of the run time of vehile
SPEED	LocationType	This keeps record of the speed of the vehicle
DIRECTION	LocationType	This keeps record of the location of the vehicle.

AutonomousVehicleController

This class is an associated class of the AutonomousVehicle that manages an enumeration representing the attributes of command status information of the vehicle. This handles commands from the SmartVehicleSystem and lets the SmartCar Service know what the appropriate state should be.

Properties

Property Name	Type	Description
routevehicle	RouteVehicleType	The variable representation of which state the vehicle is in as reported to the SmartCar Service
customerrequest	CustomerRequestType	This variable represents the state of the vehicle in response to customer requests.

Methods

Method Name	Signature	Description
getRouteVehicleType	():RouteVehicleType	Returns the enumerated property based on which state the vehicle is in through out its life cycle
getCustomerRequestType	():CustomerRequestType	Returns the enumerated properties based on the customer requests

RouteVehicleType <<enum>>

An enumeration that represents the attributes of what make up an AutonomousVehicleLocation route.

Properties

Property Name	Type	Description
PICKUP_LOCATION	RouteVehicleType	This confirms the location of the vehicle at the pick up of the Customer
DESTINATION_LOCATION	RouteVehicleType	This confirms the location of the vehicle having successfully dropped off the Customer at the destination.
SERVICE_LOCATION	RouteVehicleType	This sets the state of the vehicle as in a service location and is in need of service.
WAITING_AREA	RouteVehicleType	This sets the status of the vehicle mode as waiting.

CustomerRequestType <<enum>>

An enumeration that represents the attributes of what make up an AutonomousVehicleLocation request of a customee.

Properties

Property Name	Type	Description
COMPUTE_REQUEST	CustomerRequestType	This stores the computation of the location of the customer in relation to the current location of the vehicle
ACCEPT_REQUEST	CustomerRequestType	This sets the status of the vehicle to accept request and creates route to customer and subsequently the customer's destination
INITIATE_REQUEST	CustomerRequestType	Once completed, the vehicle sets its status to available and initiates first customer request in the queue.

Package: cscie97.asn5.vehicle

ISmartVehicleSystemAPI <<interface>>

This public interface defines the primary service contract for any classes that intend to implement these methods and act as the concrete implementation of the

SmartVehicleSystemAPI. This interface and its implementing class, collectively work as a command pattern.

Methods

Method Name	Signature	Description
executeRoute	():void	This method implements the physical movement of vehicle.
computeRoute	():void	This method returns the route given the start and end points. It can be optimized for efficiency.
computeFare	():void	This method returns the rate/miles given the start and end points. Used in determining the cost effectiveness of picking up customer from both the vehicle and customer's perspectives.
viewRideRequest	():void	This method returns requests that riders have made for that particular vehicle.
acceptRideRequest	():void	This method returns availability to a particular customer.
monitorRideRequest	():void	This method returns all requests of a particular vehicle both denied and accepted
goForService	():void	This method sets the vehicle to in need of service, alerts the shop crew and routes to the nearest service area.

SmartVehicleSystemAPI

This interface manages the concrete implementation of various commands being requested of the Smart Vehicle System for it to execute. The SmartVehicleSystemAPI registers itself as an observer of the SmartCar Service by invoking the registerObserver method. There is an assumption that this service is being handled in the cloud that can scale to a large queue of requests that are temporarily stored until the requests are either executed or denied to be cleared.

Properties

Property Name	Type	Description
commandQueue	Queue	Commands to be executed are queued in this variable

instance	SmartVehicleSystemAPI	Singleton Instance of the API; only one exists at a time.

Methods

Method Name	Signature	Description
getInstance()	SmartVehicleSystemAPI	Used to obtain a Single instance of the API for client use.
Update	(int: SmartCarServicesAPI, RouteVehicleType: routevehicle, VehicleFeedbackType:vehiclefeedback , CustomerRequestType: customerrequest): void	This method takes commands that were executed by the Vehicle and reports the state change to its place back in the SmartCar Service where it is uniquely identified.
addCommandToQueue	(Command:command):void	This method takes the various commands to be acted upon by the Vehicle and puts them in a queue.
processCommandFromQueue	():void	This method takes the various commands to be acted upon by the Vehicle and de queues them.

Command <<interface>>

Command is an interface with only one method which is execute() which is defined by the Command Pattern.

Methods

Method Name	Signature	Description
execute	():void	This method unscrupulously executes commands

CommandImplementation <<abstract>>

This is an abstract class that handles the base attributes that are common to all of the various types of commands. This is also provides an implementation for logging the commands.

Properties

Property Name	Type	Description
log	StringBuffer	This variable stores the logging of commands as they are executed.
routevehicle	RouteVehicleType	This variable points to the various enumerated types of vehicle routes
vehiclefeedback	VehicleFeedbackType	This variable points to the various enumerated types of vehicle feedback
customerrequest	CustomerRequestType	This variable points to the various enumerated types of customer requests
status	StatusType	This variable points to the various enumerated types of vehicle status

Methods

Method Name	Signature	Description
logMessage	():void	This method returns the logs of the command execution.

CommandFactory

CommandFactory is a factory class for creating different commands based on command and device information. It decouples the VehicleSystemAPI from various command instances for clear separation of concern. By delegating command instance creation process to CommandFactory, VehicleSystemAPI only needs to know about super type Command. It is also separated from the SmartCar Service in that, the commands are solely generated based on the provided services and command information by the Vehicle System.

Properties

Property Name	Type	Description
instance	CommandFactory	Singleton Instance of the CommandFactory; only one exists at a time.

Methods

Method Name	Signature	Description

getInstance	():CommandFactory	Used to obtain a Single instance of the CommandFactory.
createCommand	(smartcarservicesapi: SmartCarServicesAPI, RouteVehicleType: routevehicle, VehicleFeedbackType:vehiclefeedback, CustomerRequestType: customerrequest, StatusType:status): void	This method takes the resources of the SmartCar Service and builds the commands to be executed by the Vehicle
createRouteCommand	():Command	This creates the method that executes the vehicle route
createFareCommand	():Command	This creates the method that computes the fare for a vehicle route
createServiceCommand	():Command	This creates the method that puts the vehicle in service mode and routes to the nearest service location.
createRequestCommand	():Command	This creates the method that determines how to accept or reject request.

ExecuteRouteCommand

It extends the Command interface and implements the execute method. It is responsible for executing the controls corresponding to AutonomousVehicleController.

Methods

Method Name	**Signature**	**Description**
execute	():void	Executes the command by making API call to the AutonomousVehicleController. It calls an exception if unsuccessful.

GoForServiceCommand

It extends the Command interface and implements the execute method. It is responsible for executing the controls corresponding to AutonomousVehicleController.

Methods

Method Name	Signature	Description
execute	():void	Executes the command by making API call to the AutonomousVehicleController. It calls an exception if unsuccessful.

ComputeRouteCommand

It extends the Command interface and implements the execute method. It is responsible for executing the controls corresponding to AutonomousVehicleController.

Methods

Method Name	Signature	Description
execute	():void	Executes the command by making API call to the AutonomousVehicleController. It calls an exception if unsuccessful.

ViewRequestCommand

It extends the Command interface and implements the execute method. It is responsible for executing the controls corresponding to AutonomousVehicleController.

Methods

Method Name	Signature	Description
execute	():void	Executes the command by making API call to the AutonomousVehicleController. It calls an exception if unsuccessful.

ComputeFareCommand

It extends the Command interface and implements the execute method. It is responsible for executing the controls corresponding to AutonomousVehicleController.

Methods

Method Name	Signature	Description
execute	():void	Executes the command by making API call to the AutonomousVehicleController. It calls an exception if unsuccessful.

AcceptRequestCommand

It extends the Command interface and implements the execute method. It is responsible for executing the controls corresponding to AutonomousVehicleController.

Methods

Method Name	Signature	Description
execute	():void	Executes the command by making API call to the AutonomousVehicleController. It calls an exception if unsuccessful.

MonitorRequestCommand

It extends the Command interface and implements the execute method. It is responsible for executing the controls corresponding to AutonomousVehicleController.

Methods

Method Name	Signature	Description
execute	():void	Executes the command by making API call to the AutonomousVehicleController. It calls

		an exception if unsuccessful.

Package: cscie97.asn5.interface

IMobileApplicationAPI

This public interface defines the primary service contract for any classes that intend to implement these methods and act as the concrete implementation of the MobileApplicatoinAPI. The IMobileApplicationAPI is responsible for the layout of the front end User Interface that connects the back end services. This interface uses an EntitlementServiesAPI that implements the visitor pattern.

Methods

Method Name	Signature	Description
createRiderProfile	(AuthToken:token, int: smartCarServices, String, customerid):void	This method generates a UI interface object that links to the creation of a profile on the SmartCarServicesAPI
updateRiderProfile	(AuthToken:token, int: smartCarServices, String, customerid):void	This method modifies the UI Interface object that links the creation of a profile on the SmartCarServicesAPI
requestRide	(AuthToken:token, int: smartCarServices, String, customerid):void	This method connects a UI frame object that confirms requesting ride that talks to the SmartCarServicesAPI
checkRequstStatus	(AuthToken:token, int: smartCarServices, String, customerid):void	This method connects to a UI frame object that allows the user to track the status of the ride request once placed.
viewInvoice	(AuthToken:token, int: smartCarServices, String, customerid):void	This method connects to a UI frame object that allows you to view invoices from completed requests
shareVehicleReview	(AuthToken:token, int: smartCarServices, String, customerid):void	This method connects to a UI frame object that allows you to share vehicle reviews with other registered Rider Profiles.
viewRideOffer	(AuthToken:token, int: smartCarServices, String,	This method connects to a UI frame object that talks to the

	customerid):void	SmartCarServicesAPI which talks to the SmartVehicleServices and reports back a ride offer.
acceptRideOffer	(AuthToken:token, int: smartCarServices, String, customerid):void	This method connects to a UI frame object that signals to the Smart Vehicle that you accept the ride offer.

EntitlementServicesAPI

This implements the IEntitlementServiceAPI to provide concrete implementation for Rider usage. Observes the Singleton pattern since clients obtain the single shared instance of the object. In order to properly function, requires an initial account for usage in loading all of the items(Users, Roles, Permissions, Services); this is done in the private createUser() method which is called automatically when obtaining an instance of the EntitlementServieAPI class.

Properties

Property Name	Type	Description
admin	User	When getInstance is called, it also creates an admin which will store it here.
entitlements	Set<Entitlement>	The unique top level Entitlements contained in the Entitlement services; each Entitlement may only be declared at the top-level once but may be nested arbitrarily deep in another Entitlement
services	Set<Service>	The unique Services containe in the Entitlement services
users	Set<User>	The unique registered Users contained in the Entitlement service
Instance	EntitlementServicesAPI	Singletone instance of the EntitlementServiceAPI.

Methods

Method Name	Signature	Description
createAdmin	():void	Instantiates the EntitlementServiesAPI with a root user.
getInstance	():EntitlementServicesAPI	Returns reference to the single static instance of the

		EntitlementServiceAPI.
getUserByUserID	(userID:String):User	Helper method to retrieve a User by their ID
getUserByUserName	(userName:String):User	Helper method to retrieve a User by any of their associated usernames
getUserByAccessToken	(tokenID:String):User	Helper method to retrieve a User by the id of their AccessToken
getEntitlementByID	(entitlementID:String):Entitlement	Helper method to retrieve an Entitlement by its ID
getServiceByID	(serviceID:String):Service	Helper method to retreive a service by ID

RideRequestFormAPI

This API represents the collection of UI Frame objects that are filled out by the Rider to be communicate to the SmartCarServices/SmartVehicleSystem and processed.

Methods

Method Name	Signature	Description
requestRide	():void	This method takes the form data collected by the UI Frame and returns it to the SmartCarServices.

RideRequestManagementAPI

This API manages the various request states an represent them as UI Frame objects for the end user.

Methods

Method Name	Signature	Description
reviewOffers	():void	This method reports back from the SmartVehicleSystemAPI the vehicle availiblity and represents to the SmartCarService and to the potential Rider

		in the form of UI frame objects
selectOffers	():void	This method creates a UI frame object that represents SmartVehicleSystem Offers awaiting action from the potential Rider.
checkStatus	():void	This method creates a UI frame object that once activated, gets the state of the SmartCarServices

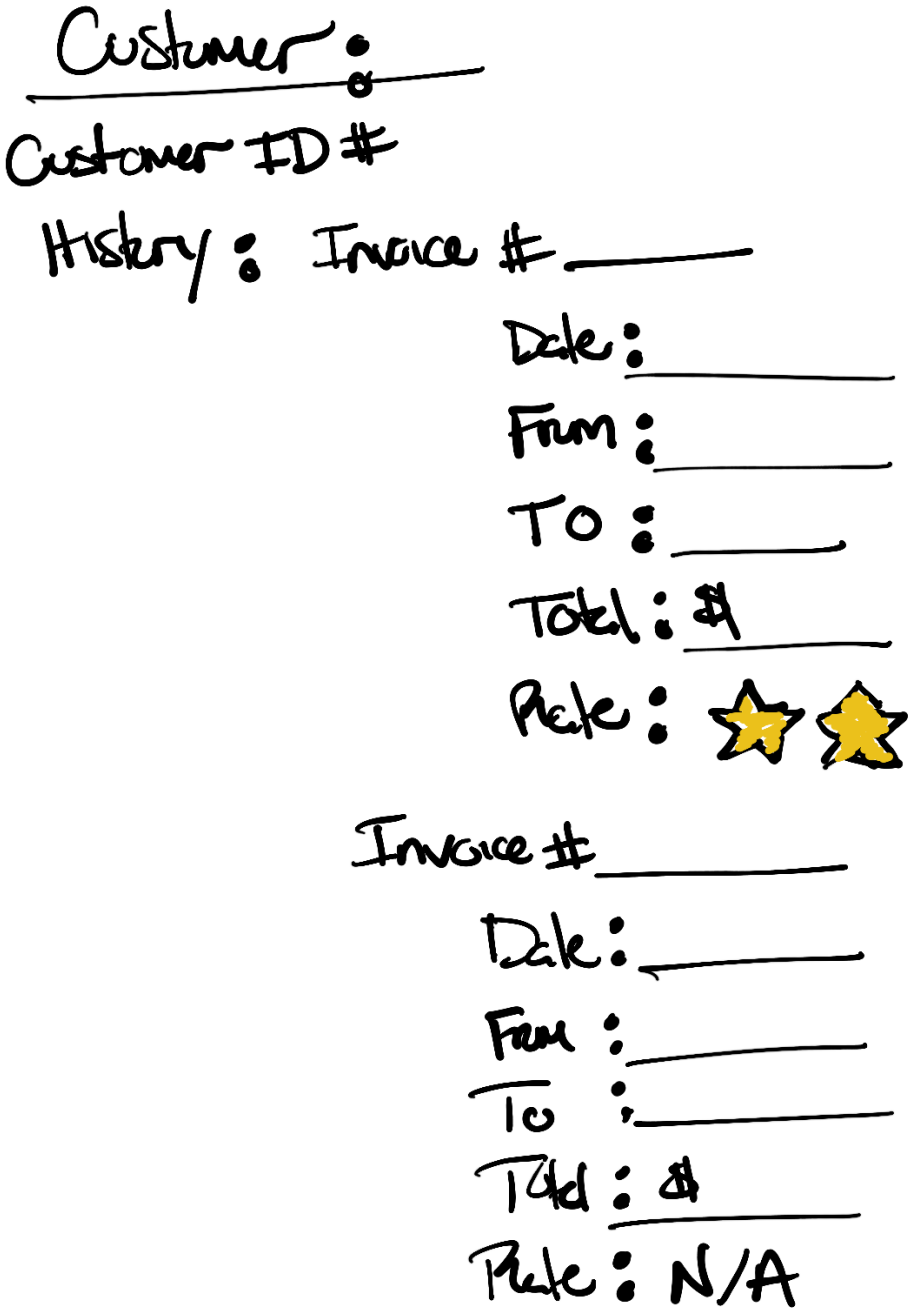
LocalStorage

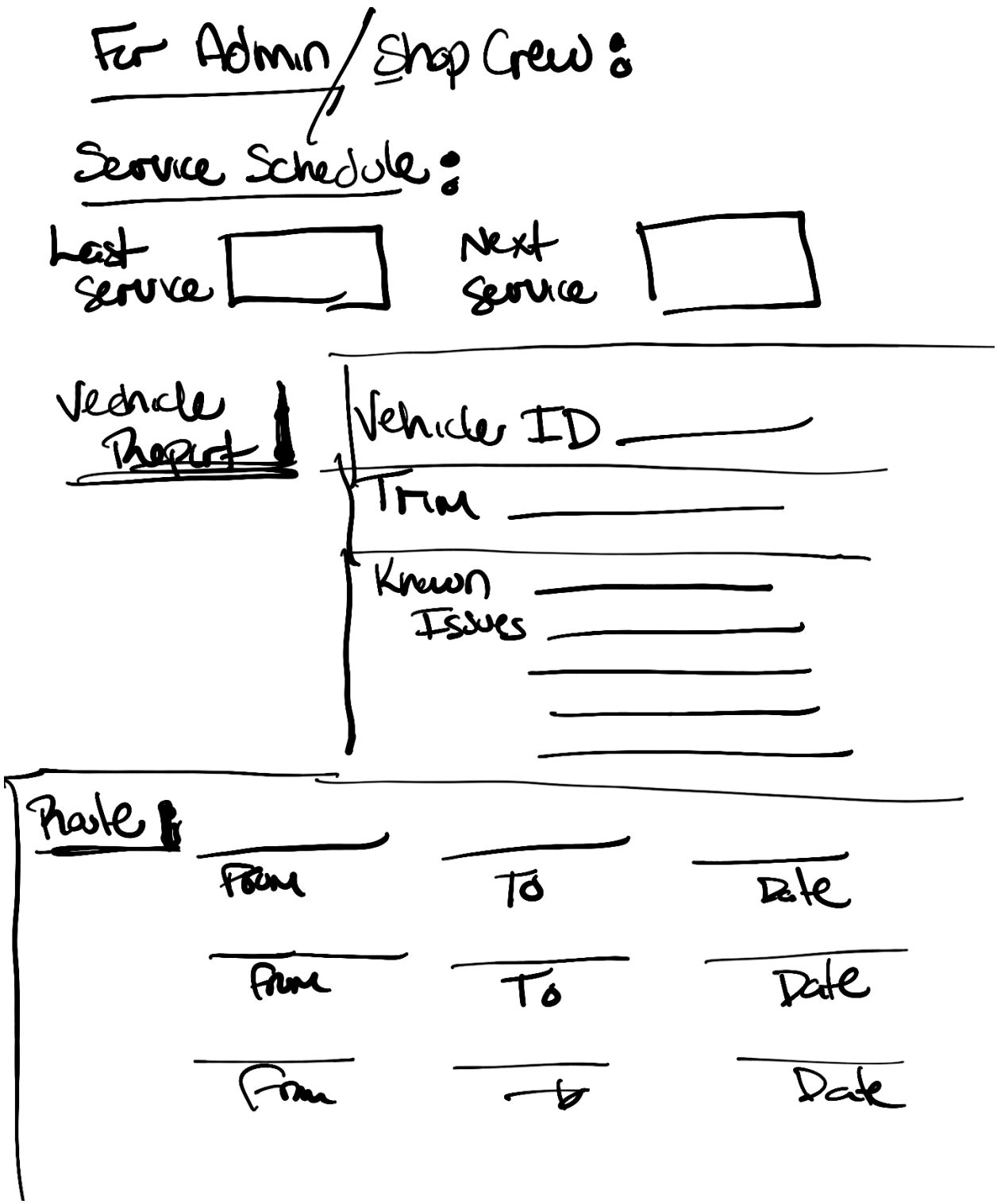
This class represents a failsafe. It attempts to act as a proxy pattern in the event that the system temporarily goes down. It stores the settings and result of user actions temporarily locally to the device and on offline cache to be written back to the system.

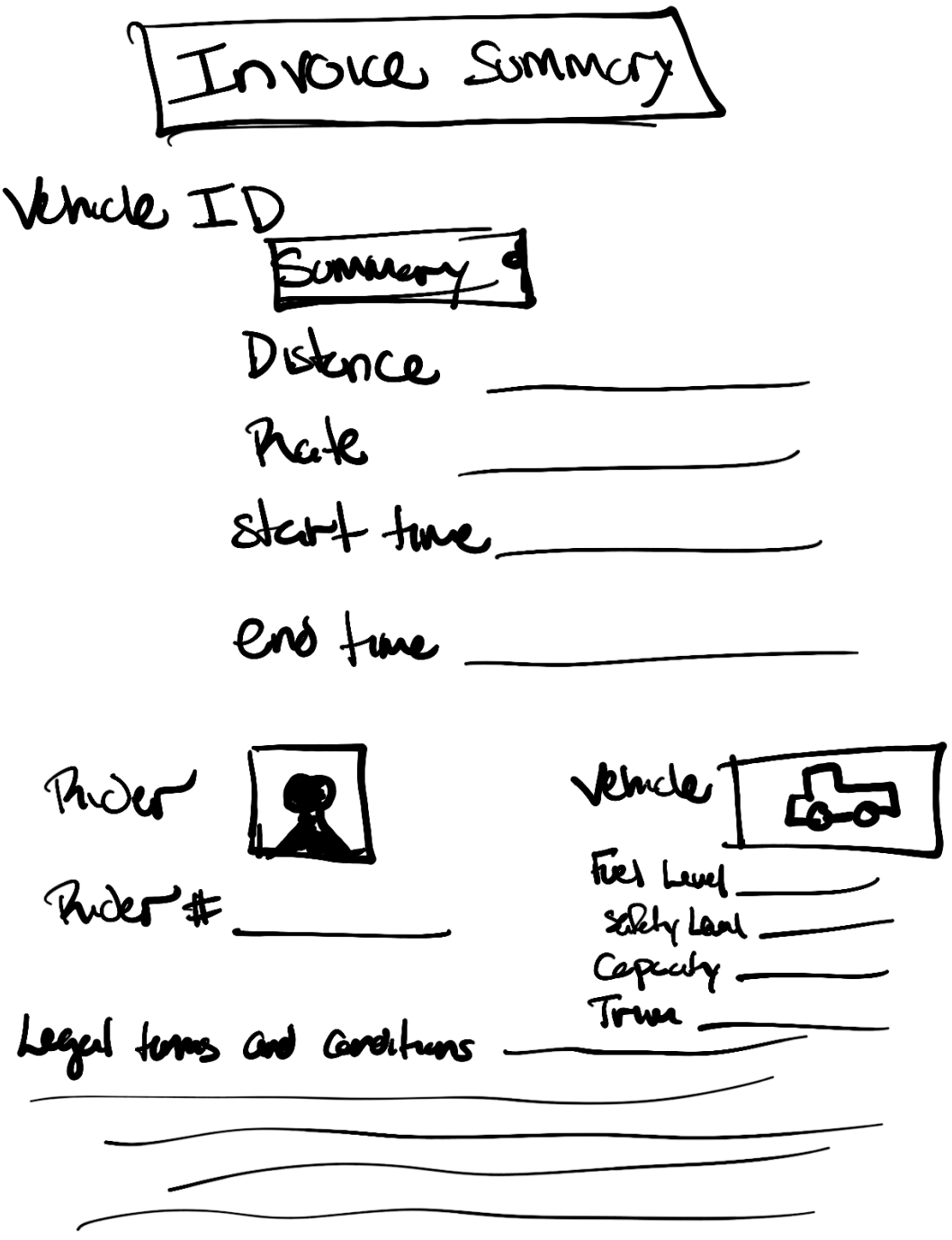
Methods

Method Name	**Signature**	**Description**
saveRideRequests	(UserRideRequets:rideRequests):void	This method stores ride requests temporarily in cache/RAM/HD depending on its size.
getRideRequests	():UserRideRequests	This method returns ride requests from cach/RAM/HD in the event that the system times out temporarily.
updateRideRequests	(UserRideRequests:rideRequests):void	This method updates the stored ride requests with newer timestamps or clears out requests that were completed or denied.
getUserPreferences	():void	This method gets user preferences after making sure that the preferences are that of the current user logged in.

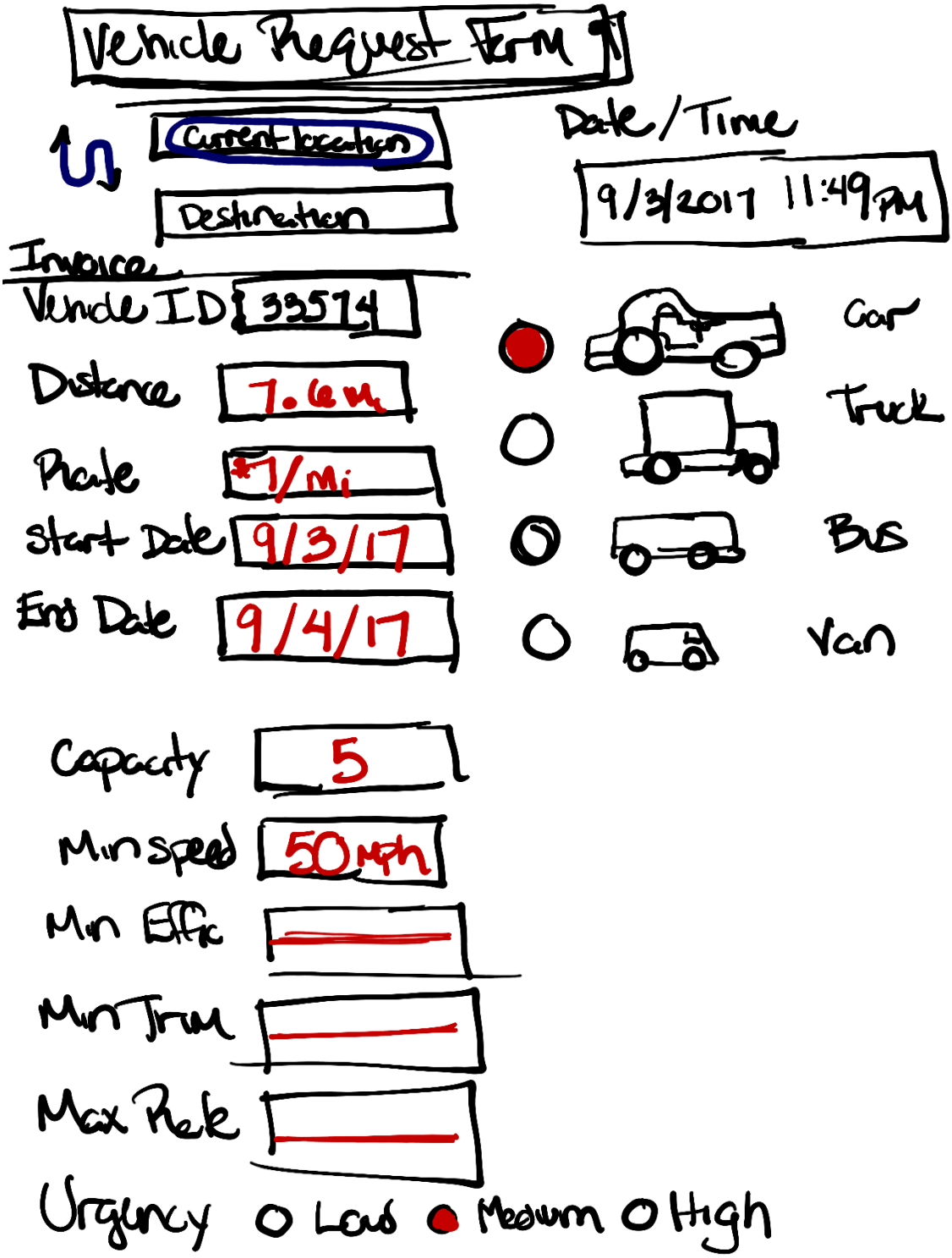
User Interface Wireframes:

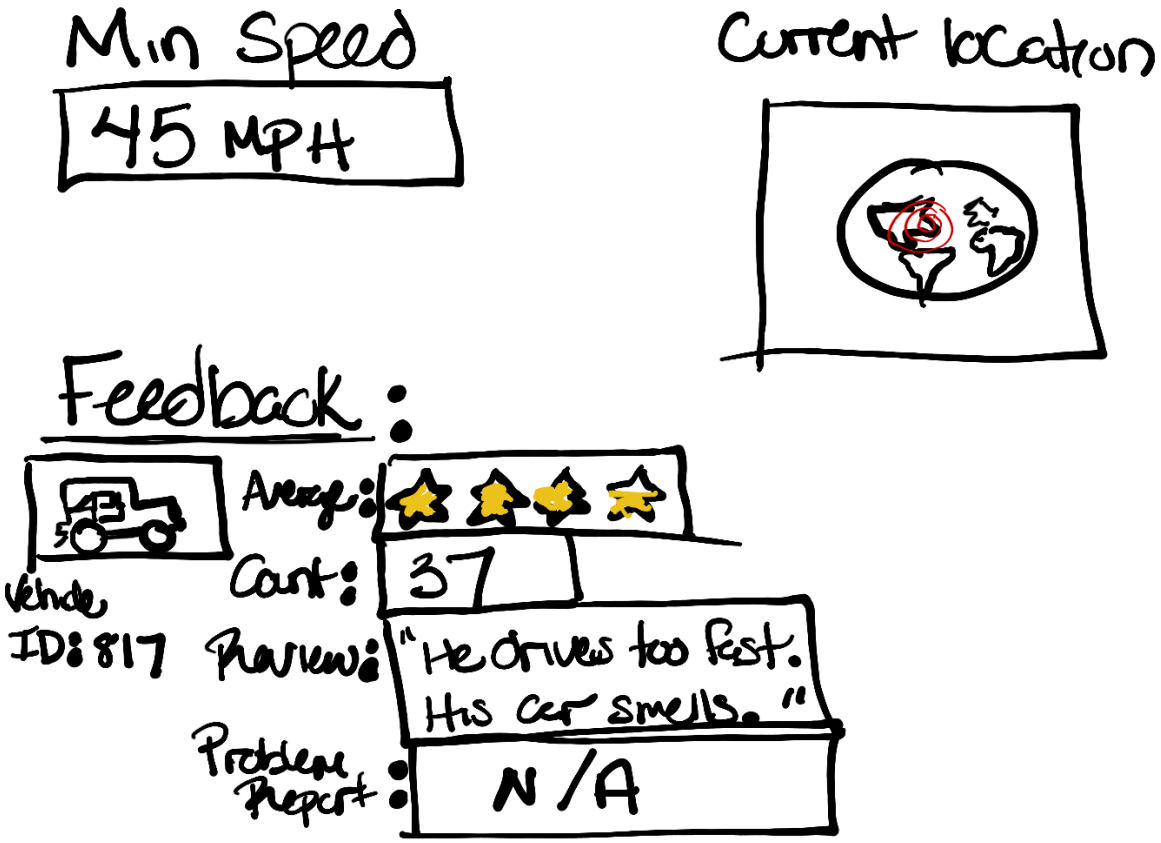


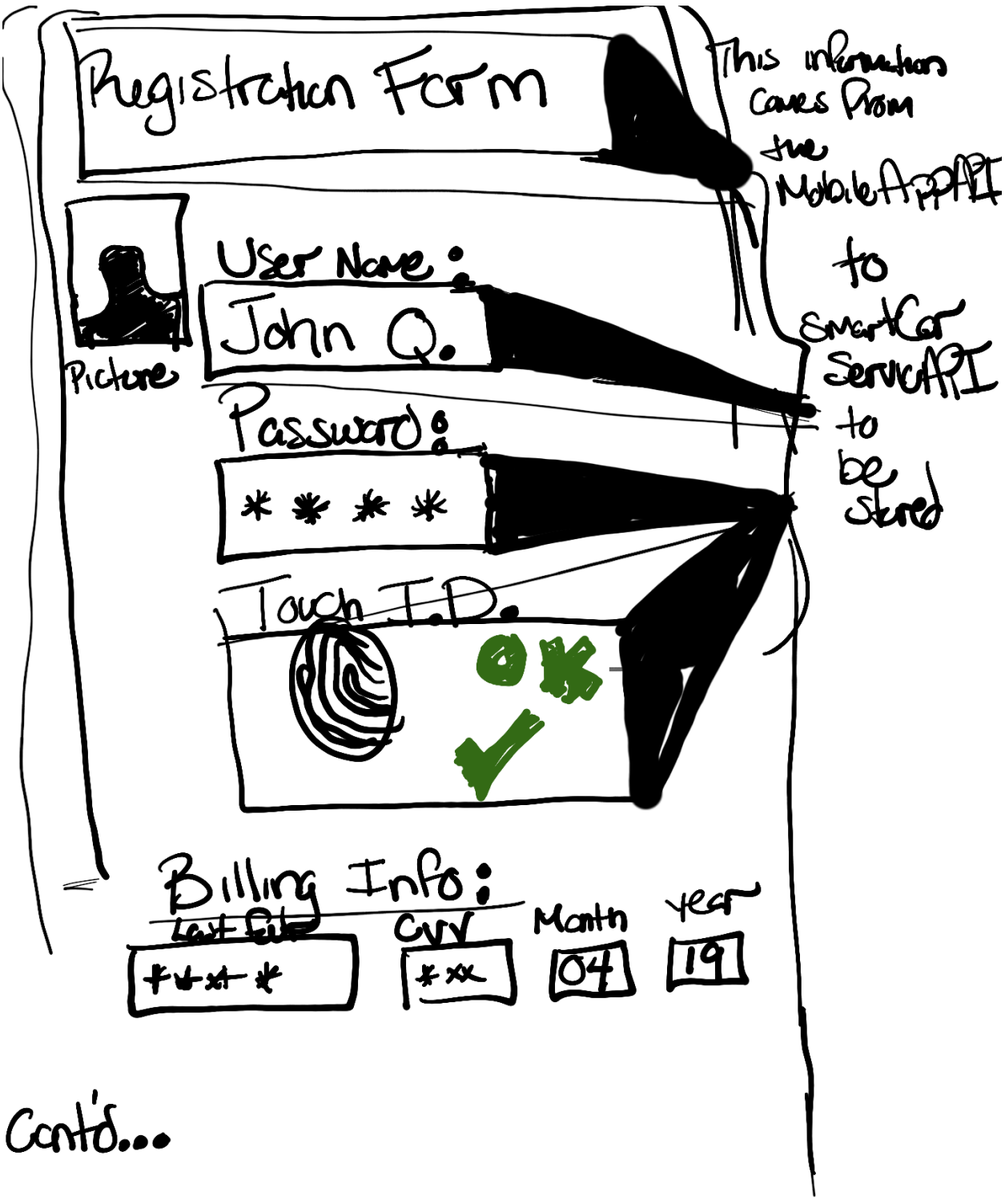

For Admin / Shop Crew:
Service Schedule:
Last Service
Next Service
Vehicle Report
Vehicle ID
Trim
Known Issues
Route
From
To
Date
From
To
Date
From
To
Date

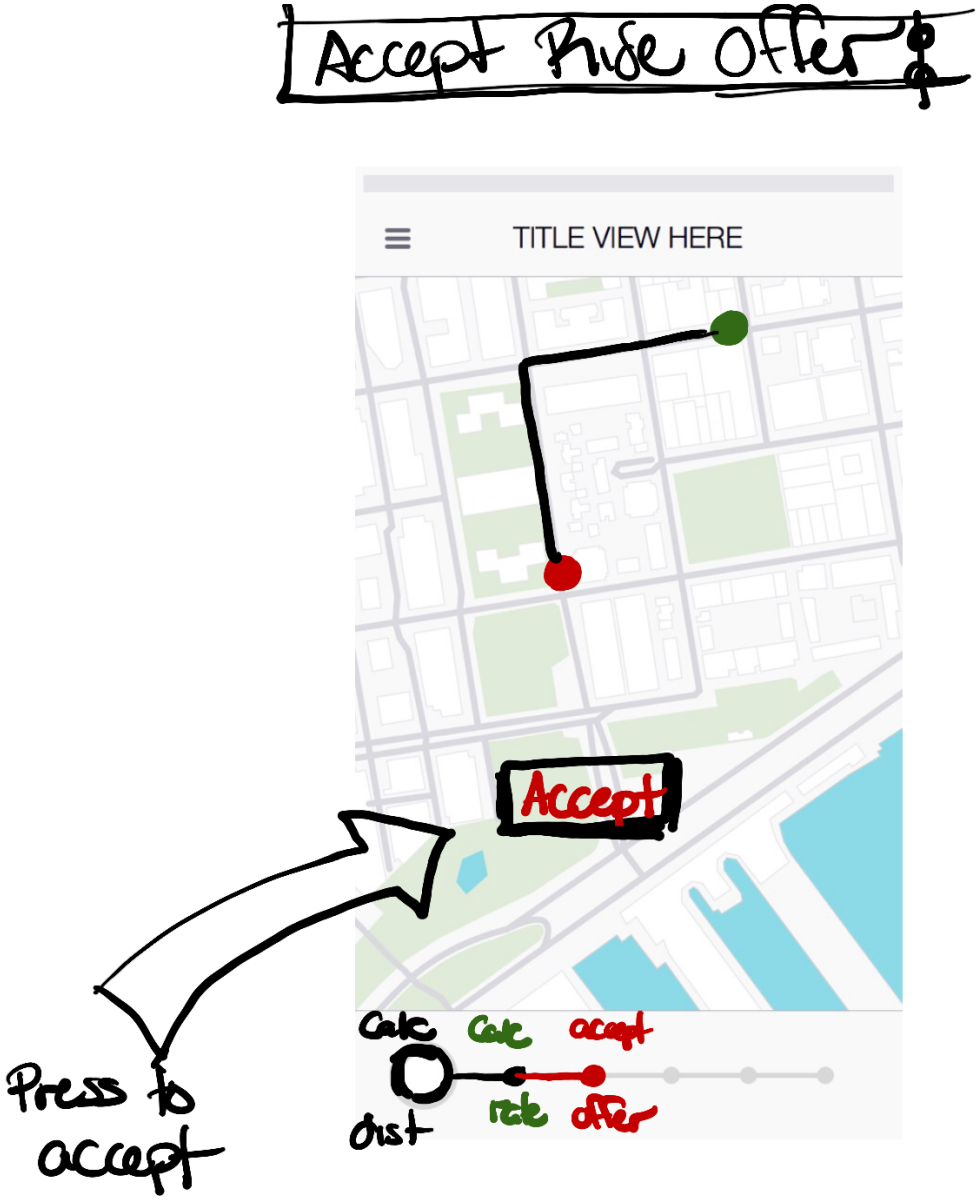

Invoice Summary
Vehicle ID
Summary
Distance
Rate
start time
end time
Rider
Rider #
Vehicle
Fuel Level
Safety Level
Capacity
Trunk
Legal terms and conditions

Vehicle request status


Vehicle Request Form
Current location
Destination
Date/Time
9/3/2017 11:49 PM
Invoice
Vehicle ID 33514
Distance 7.6 M
Plate #1/Mi
Start Date 9/3/17
End Date 9/4/17
Car
Truck
Bus
Van
Capacity 5
Min speed 50 MPH
Min Effic
Min Trim
Max Rek
Urgency Low Medium High


Min Speed
45 MPH
Current location
Feedback:
Average:
Count:
37
Vehicle ID: 817
Reviews:
"He drives too fast. His car smells."
Problem Report:
N/A


Registration Form
This information comes from the MobileAppAPI
User Name:
John Q.
Picture
to SmartCar ServiceAPI
Password:
* * * *
to be stored
Touch I.D.
OK
Billing Info:
CVV
Month
Year
04
19
Cont'd...


Accept Rise Offer
TITLE VIEW HERE
Accept
Press to accept
Calc
Calc
accept
dist
rate
offer

Implementation Details

Several classes in the SmartCars System are to be implemented as Singletons; examples are the SmartCarServiesAPI, the SmartVehicleSystemAPI, and the CommandFactory that it belonged to. SmartVehicleSystem was ripe for using the command pattern as well.

For data persistence, an Object-Relational Mapping software system should be used to directly map objects to database tables. Depending on the implementing language, if Java is selected as the implementation language and platform, Hibernate is a good choice for an ORM layer, but there are several others to choose from. For the purposes of this sample design, the objects themselves may both be directly stored in the database as well as returned to clients when requesting objects; normally the underlying objects would not be returned to clients directly, but a Data Transfer Object (DTO) would be instead

All Enumeration types should be stored in their own database tables, and when the application is loaded, they should be read from the database and populate the potential enumeration values. The requirements for this design did not call for an Entity-Relationship diagram of the intended database layout to be included, but in general, enumeration types should be short, small tables with one row per enumeration value. For objects that contain enumeration type values.

Testing

Since the application is intended to be entirely database-backed, a testing library should consist of the following:

- A set of SQL scripts that populates all of the numeration type values, each in their own database table
- A set of SQL scripts that creates all of the base object records in the database
- A set of SQL scripts that creates all of the extended properties of those base object records.
- An object loader, that will iterate over all the tables in the database, load each record, and transform that record into an object that resides in memory

Further, following Test-Driven Development (TDD) principles, a testing library should also exercise all parts of the system that interact with the database, including creating new records, updating existing records, and deleting old records. Testing should include attempts to delete non-existent records, updating non-existent records, concurrency based testing, etc.

Additionally, this system has a user front-end in the Command & Control User Interface, which will require extensive testing to verify that the forms are properly sending data to the object model, and that the objects are properly persisting in the database.

Finally, as in Assignment 4, the EntitlementServiceAPI should be tested to verify that users saved in the database are able to properly authenticate against the system, log in, log out, etc. Tests should be robust and include a variety of scenarios, including testing a logged in user attempting to access restricted interface methods with an expired accessToken, etc.

Risks

Document any risks identified during the design process.
Are there parts of the design that may not work or need to be implemented with special care or additional testing?

An admitted shortcoming in this design for the overall system is the message interchange between the MobileApplicationAPI and the SmartCarServicesAPI. I felt like I spent a lot of time working out the communication between the SmartCarServicesAPI and the SmartVehicleSystem that the MobileApplicationAPI communication was “hand waved”.

Also, as Jason pointed out in his feedback, the more I thought about my implementation of enumeration, the more I realized that the implementation would have proved to be tricky (not impossible) with some of the data manipulation.

Another risk area is the database itself; as the system is designed to be database-driven, a database failure could be catastrophic for the overall system. Therefore, a system of multiple redundancies and failover plans should be in place (database replication, remote availability, etc.).