Feature: Rental Fleet

Feature Developer: Jesus
Date Review Submitted: 3/20/2024

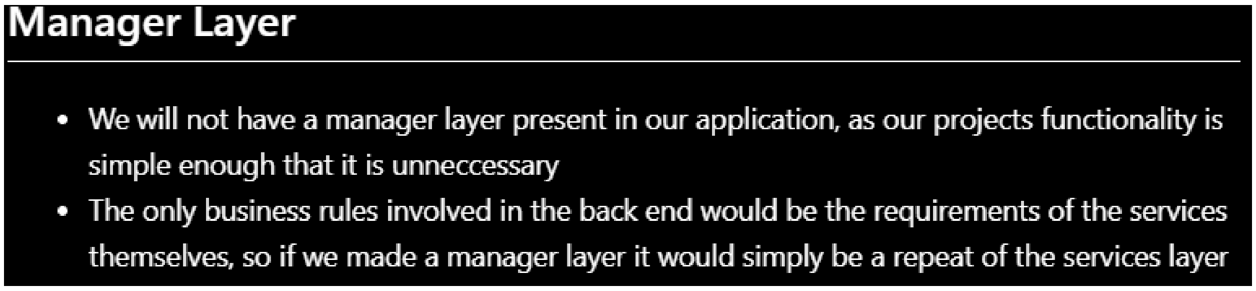
Peer Reviewer: Jason Barber
Date Review Completed: 3/21/2024

Major Positives

1. The design offers a mostly complete breakdown of each of the classes and models that will be utilized and an overview of their functionalities. The combination of both a brief written description combined with the sequence diagrams allow for a thorough explanation of the microservice readable by both business partners or developers.
2. The design accounts for Vehicle Profile Rental Statuses that have not been initialized yet by when the service is retrieving Vehicle Profile's will initialize to null and then provide a default set of values. This check will give any new vehicle profiles the default set of values that are needed. This is needed as it will force VP's to be have a default set of values.

Major Negatives

1. An issue in the design that raises concerns is the implementation of using the Vehicle Profile's Manager Layer to connect to the Rental Fleet's service. This seems to be a contradiction to later in the design which explains that

   Manager Layer

   - We will not have a manager layer present in our application, as our projects functionality is simple enough that it is unneccessary
   - The only business rules involved in the back end would be the requirements of the services themselves, so if we made a manager layer it would simply be a repeat of the services layer

   Using the Vehicle Profile Manager to call the Rental Fleet microservice, forces our system to have high coupling and low cohesion which is against the vision of our product. We want our product to have high cohesion and low coupling allowing our system to be maintained and improved on in the future. The reason this has high coupling and low cohesion is because by tying Rental Fleet to the Vehicle Profile Manager Layer any changes we must make to the Vehicle Profile Endpoints or Manager layer may force changes on Rental Fleet's methods. In turn, violates SOLID principles as now the Vehicle Profile Manager Layer serves as business checks for both Vehicle Profile and Rental Fleet regardless of how minimal Rental Fleet's business requirements are. Now this leads to not utilizing a Manager Layer at all as mentioned in the above picture, by not using a manager layer we essentially violate our High Level Design that we planned to use. Also, while the business requirements and checks aren't as numerous as Vehicle Profile or Service Log, there are checks that still need to be met as detailed in unmet requirements such as timing the service as well as some we missed in the BRD. Other examples include when setting a status for a Rental Fleet, the manager should check that the status is a valid status regardless of how it is implemented in the front - end. We want our application to be extensible and reusable and thus if we don't implement checks in the manager layer, then we potentially will be vulnerable to other attacks in the future.

2. A large flaw in the design of Rental Fleet is the absence of fail scenarios for the LLD or mention of them at all in the document. While the sequence diagram does detail what happens if a null value is found and the actions that lead afterwards, there is no detail of what happens if the data store is unavailable. In addition, other scenarios such as business requirement failures as well are not mentioned in the document but should be explained and describe what happens in the case of such failures in the system.
3. While this is a nitpick, our client has informed us to illustrate all methods in the form of UML 2.5, and the sequence diagrams do not follow this requirement.

4. Another flaw in the design is the inability to describe or illustrate the actual workings of the front - end. While the design does give the reader an idea of how the front end will be implemented with explanations, the design doesn't showcase exactly the low level working details of the system. For example, how exactly are we reaching the end points of the back- end? How are retrieving the models from local storage? If we set a status of a car, how will that be reflected in both the database and front - end? These need to be detailed in a sequence diagram for developers so that we have a plan for how the front - end should work instead of eyeballin once we get in that position.
5. Mentioned in the end point explanation that we will check authorization in this layer, but fails to mention how. Need to be explicit about the inner workings of our application.

Unmet Requirements

- **FM - 1**
    - **Pass Requirements:**
        - **Dashboard must generate and add all Rental Fleets Vehicle profiles to view within 3 seconds**
        - Each Vehicle profile must display the following 6 parameters:
            - Last Recorded Mileage
            - Approximate Due Date of next Maintenance Event
- **FM - 2**
    - **Pass Requirements:**

- If there are more than 10 vehicles that contain the keyword that the user provided, the application will display 10 at a time and each time the user requests to see more, the application will load 10 more posts at a time.
- Each Vehicle profile must display the following 6 parameters:
  - Last Recorded Mileage
  - Approximate Due Date of next Maintenance Event

Design Recommendations

1. The easiest solution to resolving the first major negative of the Peer Review is to implement a Manager Layer to not only follow the design of our application but also check business logic for the Services Layer. In turn, this forces us to follow SOLID principles which are required by our Product manager, as well as follow the Single Responsibility Principle.
2. In order to describe and showcase when our system encounters errors, create sequence diagrams on how our system handles errors with fail scenarios describing the event. Further explanation would be a plus for both developers and clients if needed.
3. Make all sequence diagrams follow the UML standard set by our client to follow their directions and promote satisfaction.
4. Showcase the inner workings of the front - end of the sequence diagrams or if too complex make entirely new sequence diagrams for the front end portion. Like what our client said, we should be spending 80% designing and 20% coding. By planning out our design, we reduce the need to go back and have to modify existing code and such.

5. Explicitly detail on the sequence diagrams of how we are checking authorization for the application and the method calls we need to do in order to check it as well as being in the correct UML format.

Test Recommendations

1. While other functionalities may not be finished such as Service Log, testing to make all displays function correctly and are able to display data is crucial to the feature working efficiently enough for customers satisfaction.
2. Due to not having a manager layer at the time of creating the LLD, testing to make sure we have a check for a timer in the manager layers is important as well, so that we are not trying to connect to a database that is down.
3. Need to test when a Rental Fleet user has one Vehicle Profile or 50 Vehicle Profiles and all the numbers in between to make sure that the feature will display all the Vehicle Profiles correctly and in the correct status bar.
4. Needs to be tested for what happens when a user puts in an incorrect search make, model, or manufacturer and if the system reacts accordingly.