VILNIUS UNIVERSITY

MATHEMATICS AND INFORMATICS FACULTY

SOFTWARE ENGINEERING

Locals to Locals

Second Laboratory Work

Software Engineering II

Submitted by:

Arnas Rimkus,

Deividas Kučinskas,

Matas Lazdauskas,

Jaroslav Kochanovskis,

Gytis Bečalis

Supervisor: Assist. Dr. Vytautas Valaitis

Vilnius 2021

Summary

As we continue to learn software engineering principles and apply them in development of an actual application, in this paper we design the system and the required changes. While in the first work we mostly analyzed business and its processes this time we will mainly pay attention to the design of the existing system and the required changes. For this laboratory work, we have been given the following tasks:

1. Use UML 4+1 framework for organizing the architecture document.
2. Implement the planned changes in the system.
3. Introduce a CI/CD process.

The expected output is a system architecture document, implemented changes and a deployed CI/CD process.

Table of Contents

1. Context

For every system to be successful it must be developed with the intention of solving some sort of a real-world problem in mind. Even the most planned out and well-made software is useless if it does not solve the problem required. In this part we analyze what kind of problem is being solved and how it is intended to be solved.

1.1 Goal of the System

Connect local business with a larger clientele base.

1.1.1 The problem

In most cases it is too expensive or too difficult for small local sellers to own and manage a website. Even though there is the possibility to advertise on social media, a very large information flow and advertisements form large companies makes it hard to be noticed.

1.1.2 Solution

To solve the mentioned problem system that allows vendors to create their service profiles and users to browse them was created. The core attributes of allowing users to choose vendor are as follows:

1. Details of products and work hours,
2. Exact vendor location,
3. Reviews of other users.

Reviews are important since it helps to measure reliability and quality of services and products. Another important feature to mention is the ability to browse services that are nearby the user.

1.1.3 Main User Goals

User base could be split into two main groups:

1. Buyers
2. Vendors

The main goal for buyers is to find nearest services that sell/provide high quality production/services. The second group is mostly interested in advertising services and attracting customers.

1.2 Planned Changes

To further develop and increase the functionality of the existing system we were given the task of implementing several changes and features. The improvements consist of expanding vendor’s service options by implementing listings into services and by creating basic moderation features.

1.2.1 Change List

As more detailed requirements were provided in the first paper, we only include initial stakeholder’s requirements:

1. Every service should be able to add product listings.
2. Create admin and user roles:
3. Admin can delete services, reviews, and any other inappropriate content.
4. Registered users can report services, reviews, and any other inappropriate content.
5. Admins to have their own separate menu on the site, where they can handle user reports.

1.2.2 Impact of Changes

The newly added product lists will allow vendors to include structured lists of their products and services as well as prices. This will help clients to better determine whether it is the service they want and how much will their target product cost. Roles as well as reports enable the more in-depth moderation of posts, reviews, and other user generated content. This will stop fake advertising, misleading reviews as well as inappropriate content.

1.3 Current System Analysis

Since we had no part in the creation of the original system, it was important to analyze the system in its current state and find its limitations from scratch so we could clarify and document existing problems and consider the limits and boundaries when designing changes.

1.3.1 System Environment

The main web-application server will be run on a Windows machine using IIS Express. The database will be run from a separate Linux server. Clients will be able to connect through any type of modern browser. Support for older browsers (e.g., Internet Explorer) is not included.

1.3.2 Tools and Technologies

The main development tools and technologies, which are the .NET Core software framework and ASP.NET server-side web-application framework was determined by the original university exercise requirements. They fit the system at hand well, allowing for easy and fast web-application development. Another requirement was using a relational database, specifically a database management system based on SQL. The database management system chosen by the original developers was PostgreSQL, which is free and open source, also is a good fit for our system.

Other tools however were given greater freedom of choice. An important decision was to use the MVC architecture for the web-application, which while not being the most modern choice is a well-tested and proven method of creating web-applications.

Overall, the tools and technologies are well chosen for the system being developed.

1.3.3 Existing Problems

During the testing of the system and acknowledgement of source code we compiled a (non-exhaustive) list of existing boundaries and problems:

1. Mixing of CSS usage types (inline and external)
2. Code is not commented nor documented in any way,
3. Exceptions when certain database tables or specific to logged in user parts are empty,
4. Index page banner repeats instead of filling whole space,
5. After changing language to Lithuanian some FAQ sections stays in English,
6. Limited entry of work hours (only from 9 am to 8:30 pm).
7. Choosing an address when creating a service is faulty (Cannot load google maps correctly, Geocoder is not accepting requests).
8. Password recovery email does not seem to work (crashes website).
9. You can subscribe to your own account newsletter.

10. Changing password takes no input validation.
11. Missing many alerts when entering empty data
12. Missing alerts for which field are necessary when filling in forms.
13. A single user can leave multiple reviews for the same service, nor can the user delete or edit his review.

1.4 Development Environment

Version control systems, source code management systems, source code hosting environments and many other facets that are part of software configuration management play a major role in any modern software development project, in general improving the quality of the product greatly. This is especially important in our case, since our team is working fully remotely, which hampers communication and makes it more difficult to keep track of the whole project. As such we decided to briefly explain how our work environment looks.

Our version control system is Git, which we chose because of its widespread use. The source code is hosted on GitHub because we find it easier to use than other repository management tools as well as our whole team being familiar with it. Following best practices, we do code reviews and as a rule of thumb do not let code into production without review from other team members. We have also implemented a simple CI/CD process, to automatically check for style and build errors.

2. 4+1 Architectural View Model

To better understand the system, its current and future implementation as well as describe sequences of interactions between objects and between processes we composed a set of diagrams using UML 4+1 architectural view model. The four views of the model are logical, development, process, and physical view. In addition, selected use cases or scenarios are used to illustrate the architecture.

2.1Logical view

Logical view describes the functionality that the system provides to end-users. This is achieved via these diagrams:

1. Class diagrams,
2. Object diagram,
3. Collaboration diagram,
4. State machine diagrams

Each of these diagrams has a separate section in which diagrams itself and descriptions are provided.

2.1.1 Class Diagrams

The class diagram before additions shown in Figure 1, displays the general structure of the main code parts before any additions, in its current state. This diagram helps to get a better understanding of how the code is structured. We can see that the code is structured in MVC pattern, with controllers used to execute methods related to classes. This structure is something worth keeping in mind, as deviating too far from it could result in confusion in the code. This diagram can be used as a reference point to build upon the code because it contains the main architecture of the code.

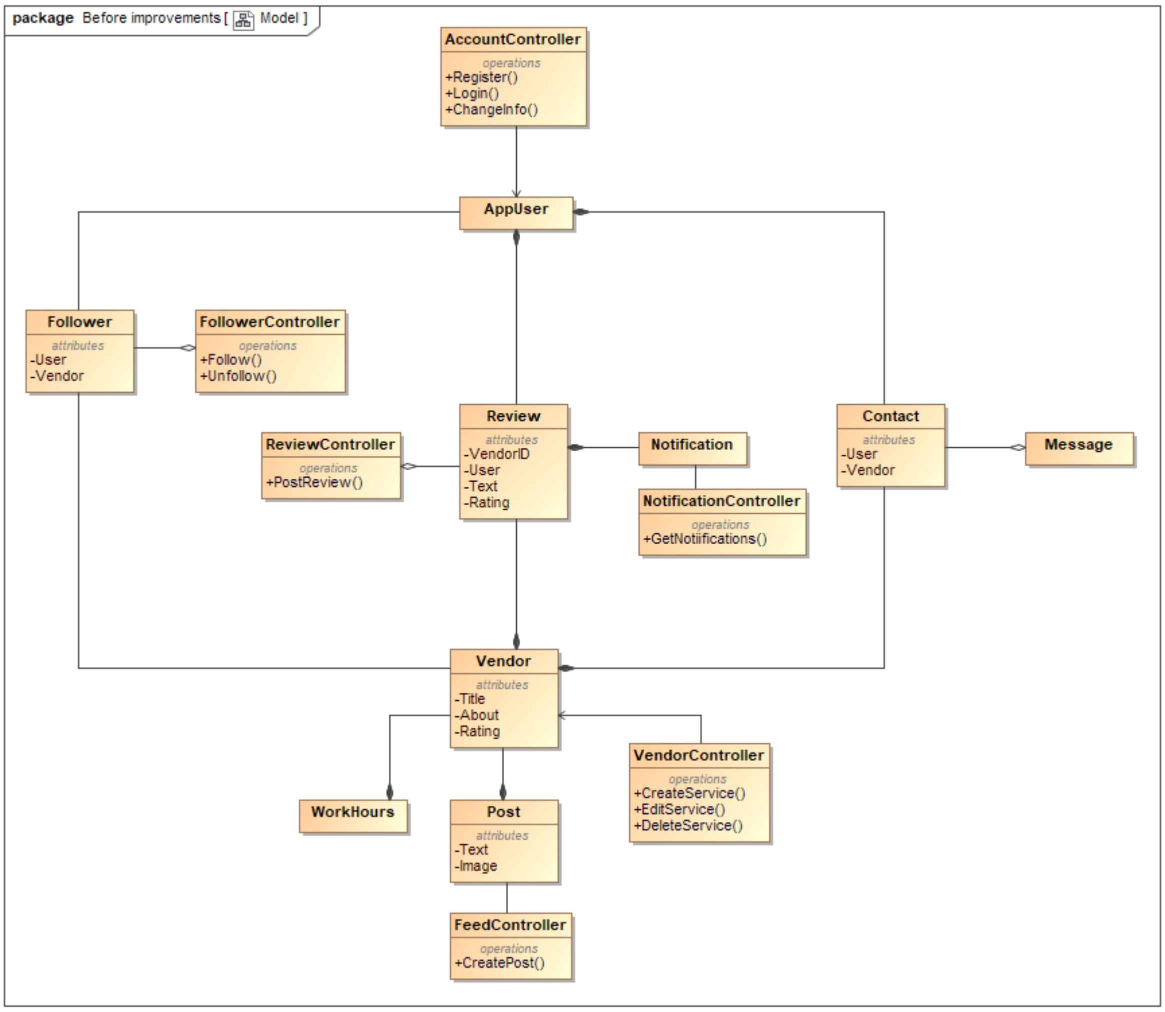

Figure 1. Class Diagram Before Additions

The class diagram after additions shown in Figure 2 displays how the general functionality of the system will change after implementing the planned additions. For example, the “Admin” and “Listing” classes are crucial to fulfill requirements. Their required methods and/or attributes are shown in a general view, enough to have a blueprint, and are subject to change in the coding process. The main takeaway is that modifications for adding the requirements should be small, and so the coding process will mostly consist of adding new code, instead of altering current code.

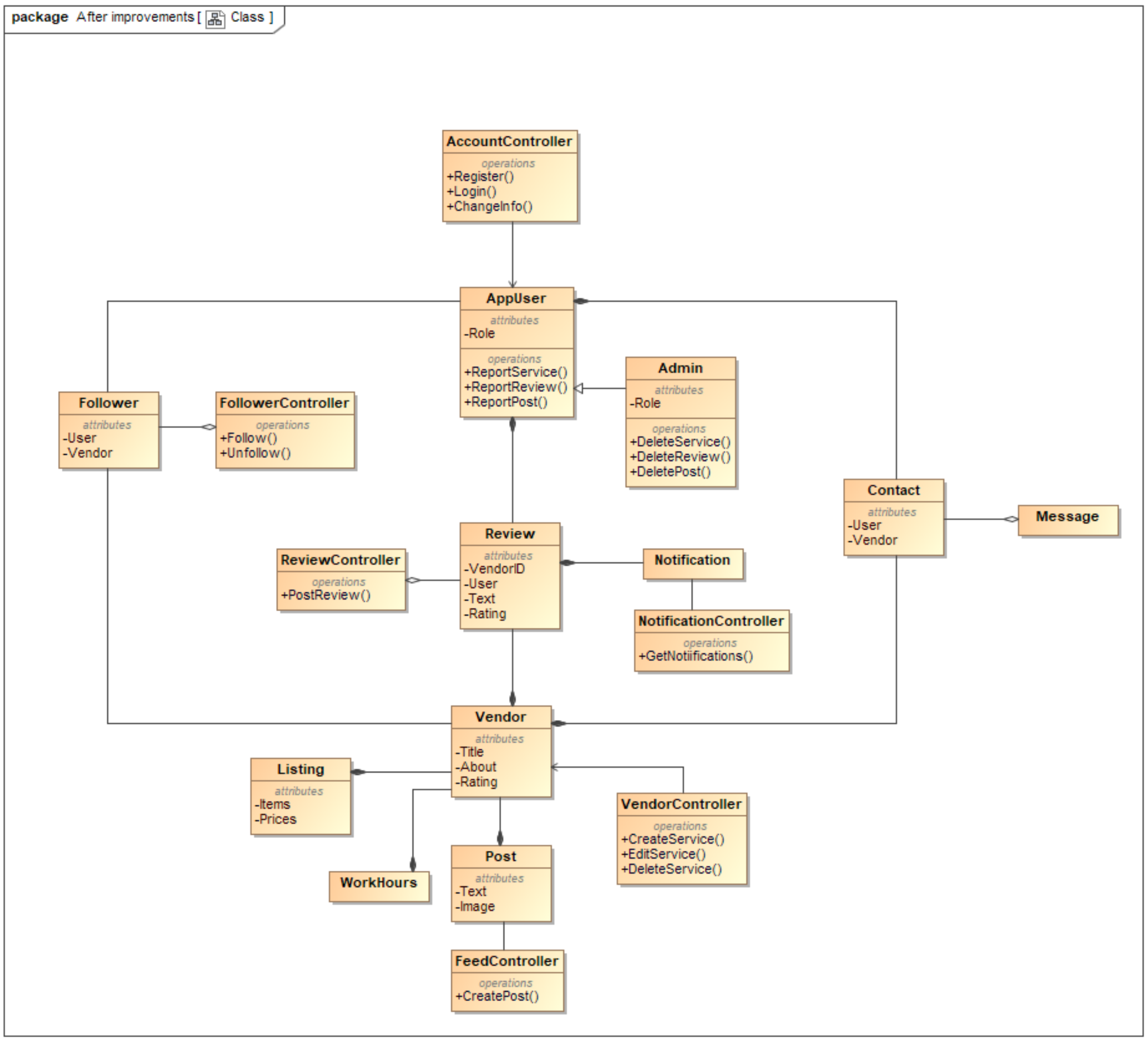

Figure 2. Class Diagram After Additions

2.1.2 Object Diagram

The object diagram shown in Figure 3 illustrates a static point in time with a "Vendor" and its related objects filled with some demonstrational data, to show a class diagram instance in accordance with the planned changes. Something to focus on here is the "Listing" object because it is yet to be integrated. Essentially, the diagram is one example of how the system's data could look with completed implementations after use. This can be used as a type of simple test after the coding process. If this situation is reproducible in the app, that means the listing requirement is complete and functional.

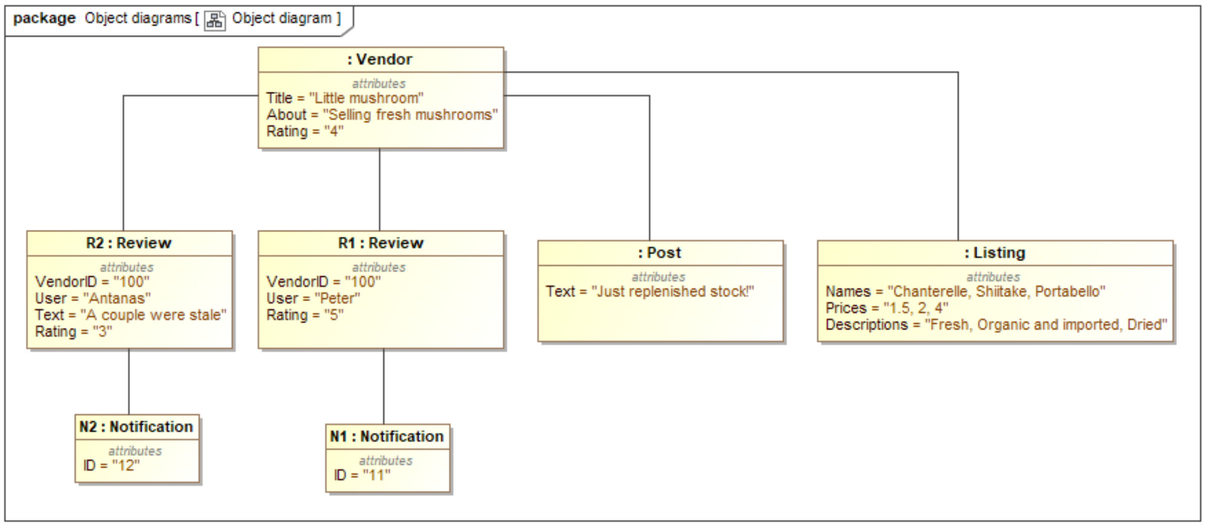

Figure 3. Object Diagram

2.1.3 Collaboration Diagrams

Collaboration diagrams are used here to illustrate how the system works and how objects communicate. The diagrams help clarify the roles of objects in scope of the new functionality needed for requirements and assists in understanding the needed additions for sending messages in the code. Figure 4 showcases listing creation communication while Figure 5 explains report handling communication. In both cases, the new changes require communication with the database, as such they are not merely cosmetic changes to the system.

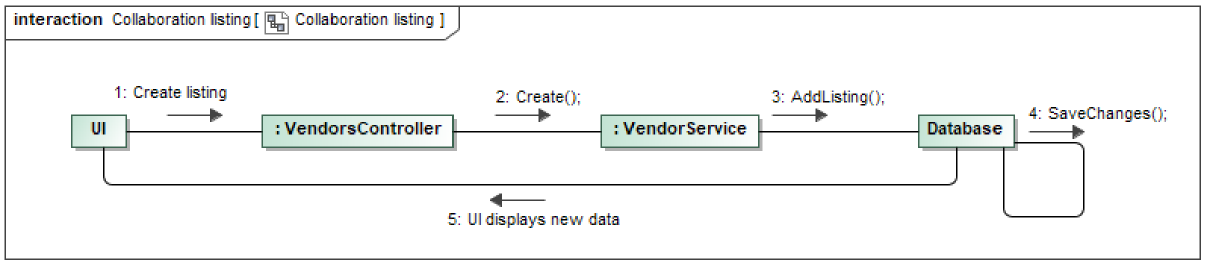

Figure 4. Listing Creation Collaboration Diagram

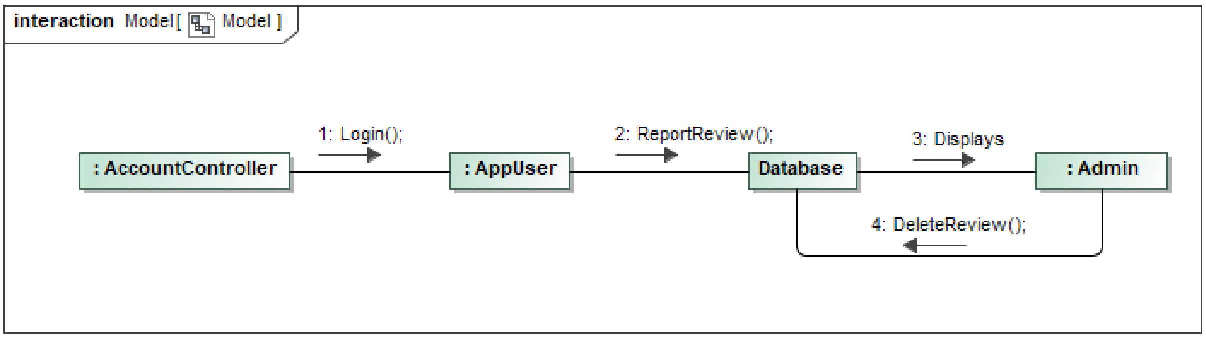

Figure 5. Report Handling Collaboration Diagram

2.1.4 State Machine Diagrams

We used UML state machine diagrams to determine which entities of the system have states and how these states change and react. In Figure 6 we can see user states. Users can either be *authorized* (by registration or logging in) or *unauthorized*. The most important part is that *authorized* users can create a service. *Authorized* user that creates a new or already has existing service (-es) is a *vendor*. Since the number of services is unlimited, users can have multiple. The user loses *vendor* state when he deletes his last service (in some cases the service could be deleted by an administrator).

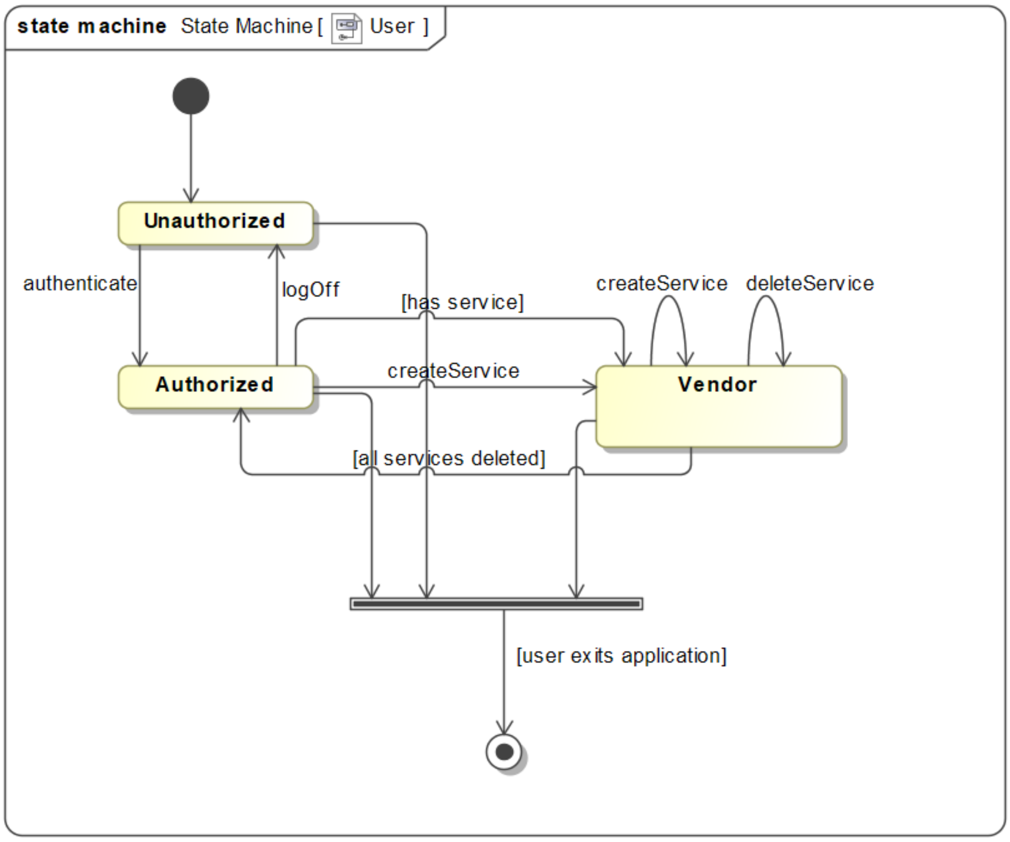

Figure 6. User State Machine

Another part of the system that has states is services. Current states of service are displayed in Figure 7. Currently it only has state published and while in this state can be updated. Later, service can be deleted. The implementation of the new feature that adds product lists to the services changes the state diagram. This can be seen in Figure 8. As it is shown services can be created with listings or the listings can be added after the service has been published. Product listings can also be deleted, after which the service becomes *published*.

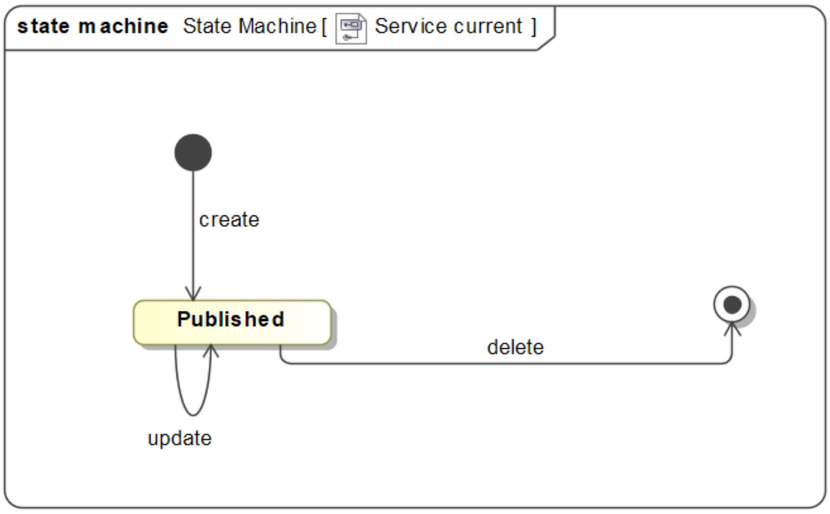

Figure 7. Service State Machine Before Changes

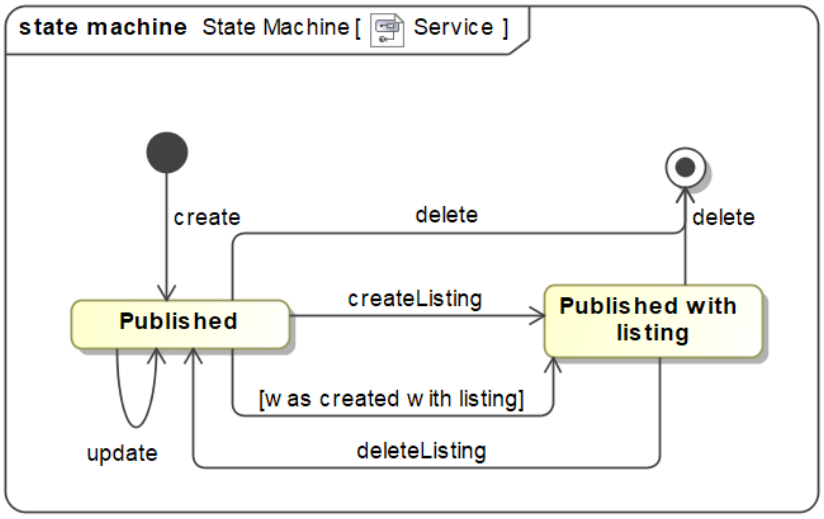

Figure 8. Service State Machine After Changes

The last state machine diagram shown in Figure 9 illustrates the reporting feature that is to be implemented by our team. When the user reports any user-generated publication it becomes an unreviewed report. Later administrators can discard (delete) or accept the report. After the report has been approved action can be taken and after that the report should be archived. When it is archived, the report can be retrieved in the future if needed or deleted either manually or after a certain amount of time passes.

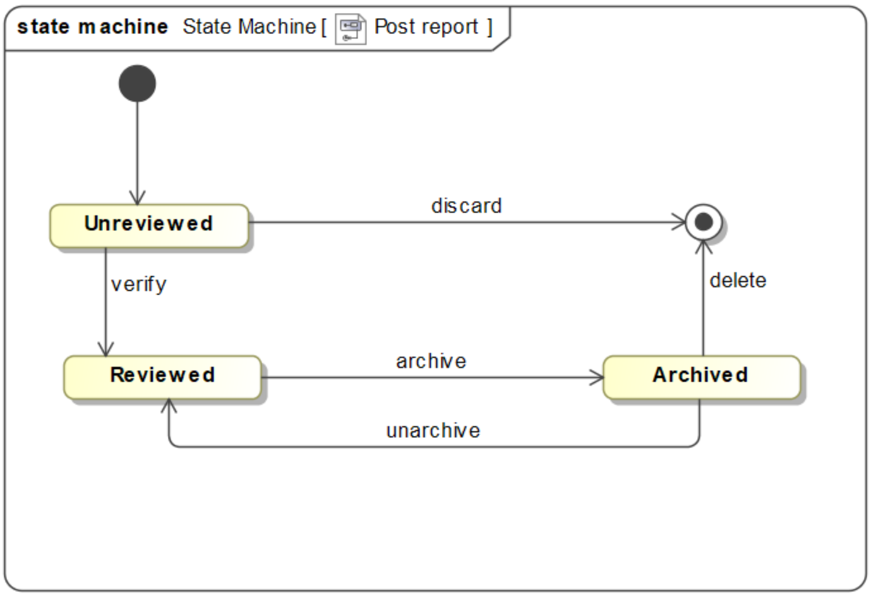

Figure 9. Report State Machine

2.2Development View

The development view illustrates a system from a programmer’s perspective and is concerned with software management. The main types of diagrams associated with this view are component and package diagrams.

2.2.1 Component Diagram

This section will illustrate the implementation and components of the system. The general view of the project component diagram is provided in Figure 10. All actors are using different UIs, but all these UIs are connected to one system. The system shown is after the implementation of the changes. Current diagram would be the same but without the *Admin UI* and *Admin* as an actor.

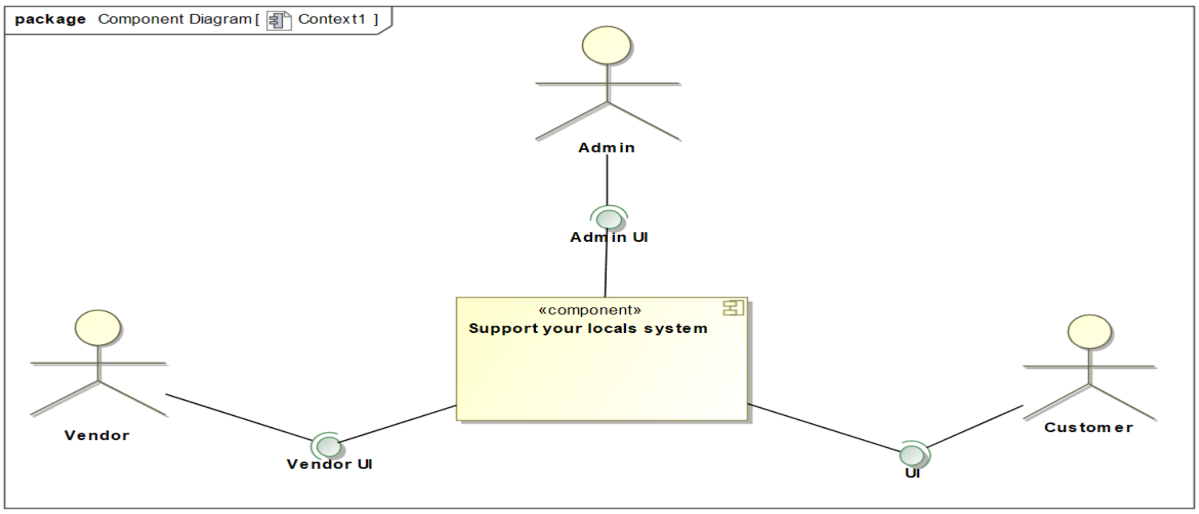
package Component Diagram [Context1]
Admin
Admin UI
«component»
Support your locals system
Vendor
Vendor UI
Customer
UI

Figure 10. General Component Diagram

Decomposition of Support your Locals system component is provided in Figure 11 and Figure 12. In Figure 11 we can see what inner components the system consists of before our modifications. The main thing is Controller that handles all this system. From the OpenStreetMap API we get a map view.

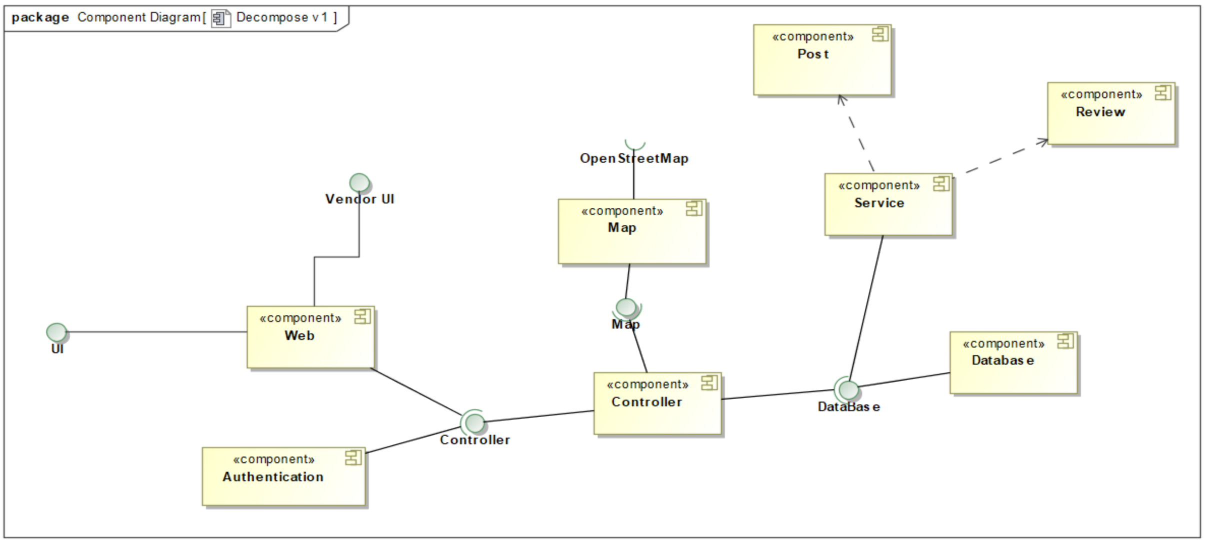

Figure 11. Decomposition Diagram Before Changes

In Figure 12 we can see how the system decomposition looks after our modifications. As the stakeholder wanted, we are adding the Admin role and its UI. Another component which will be saved in the database is *Report*.

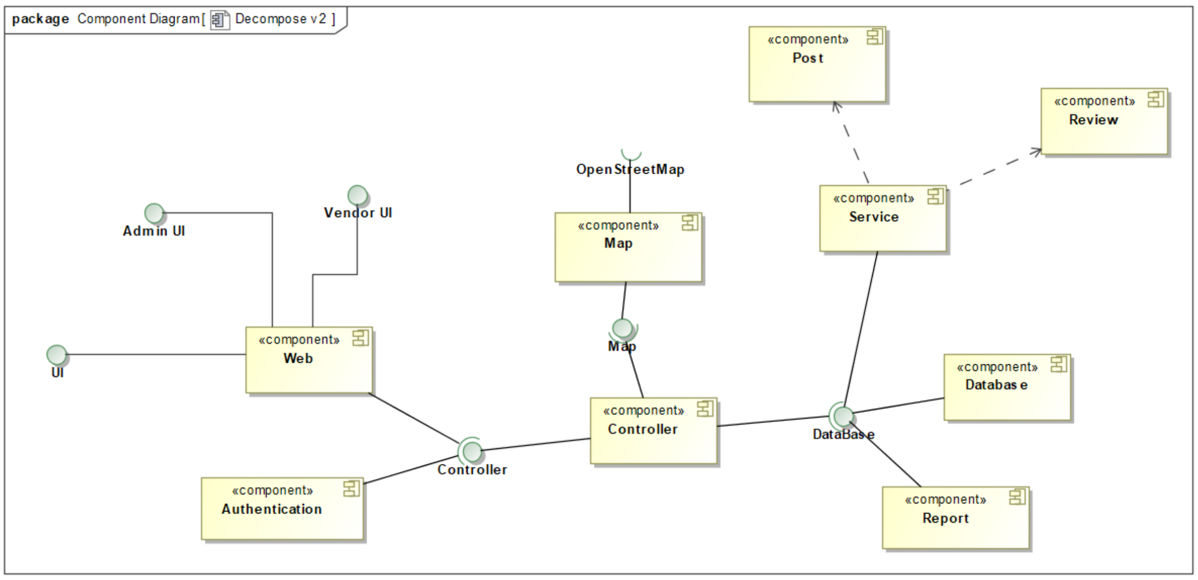

Figure 12. Decomposition Diagram After Changes

Information Flow diagram displayed in Figure 13 shows how information transits from a source to a receiver.

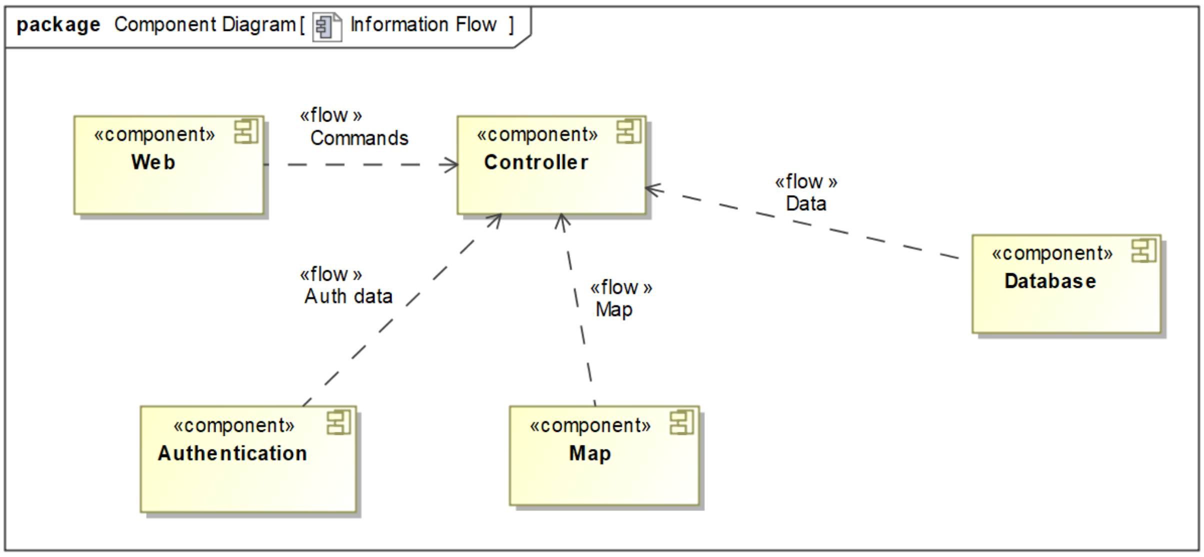

Figure 13. Information Flow Diagram

2.2.2 Package Diagram

Package Diagram in Figure 14 reveals the dependencies between the packages that make up a model.

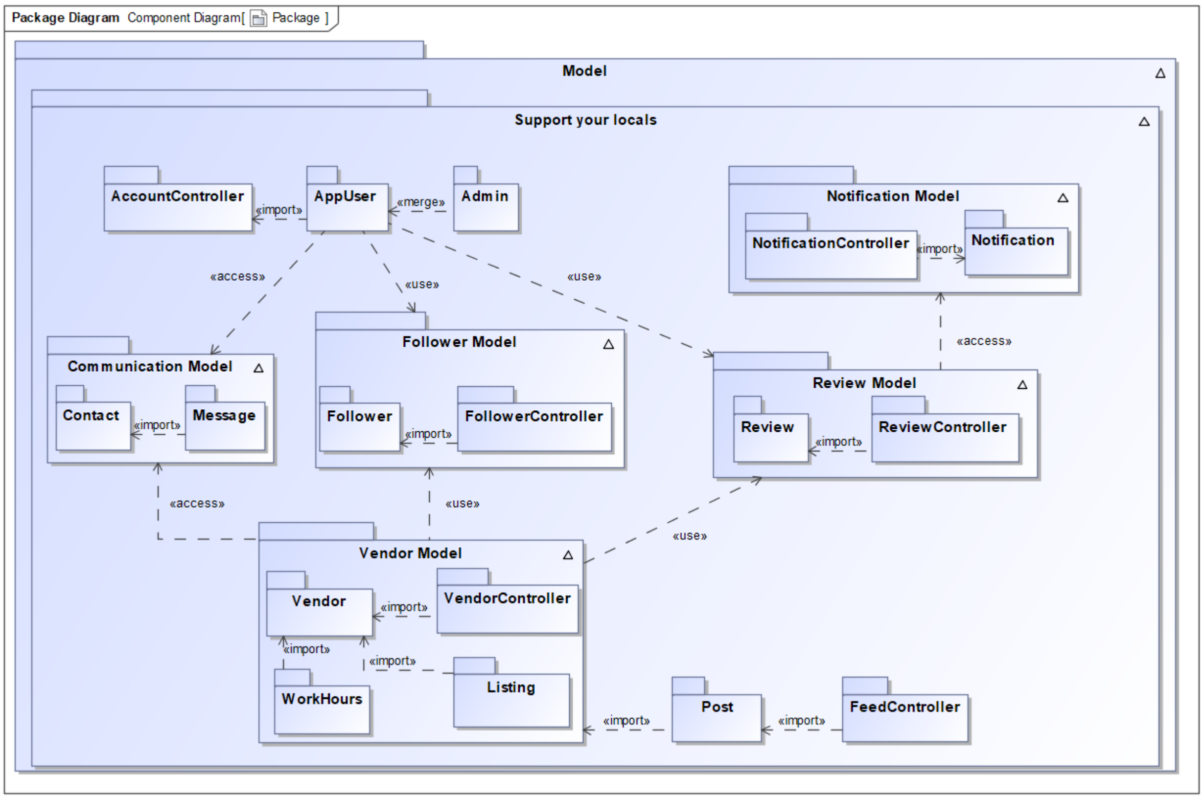

Figure 14. Package Diagram

2.3Process View

Process view illustrates and explains the systems processes. The focus is on their communication and synchronization. The main diagrams which help to understand these processes are activity diagrams and sequence diagrams.

2.3.1 Activity Diagrams

Vendor's service management process before changes is covered in Figure 15. This diagram showcases existing vendor related processes. Vendors can currently set up services and edit their details or delete them.

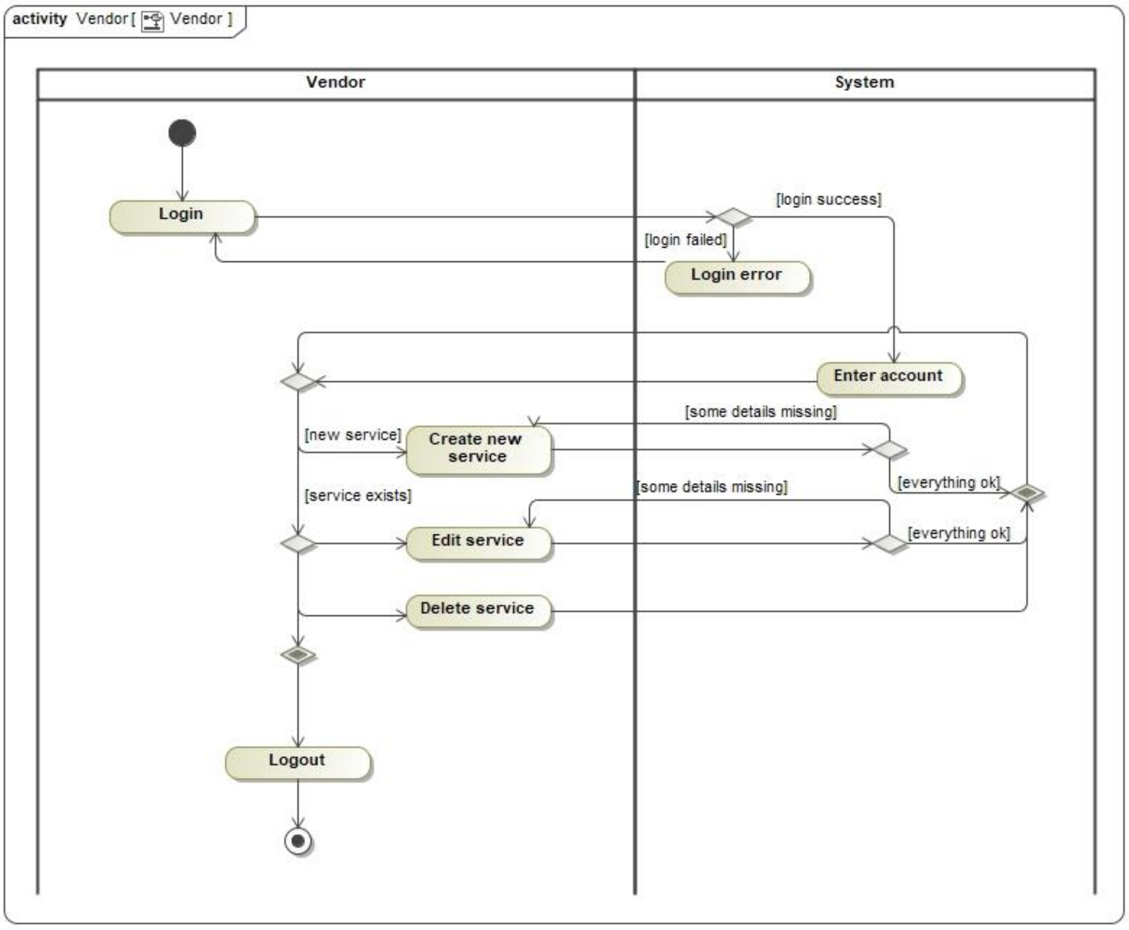

Figure 15. Vendor Activity Diagram Before Changes

After implementation of required features as we can see in Figure 16 vendor is able to add and edit listings during processes of service creation and modification if so desired.

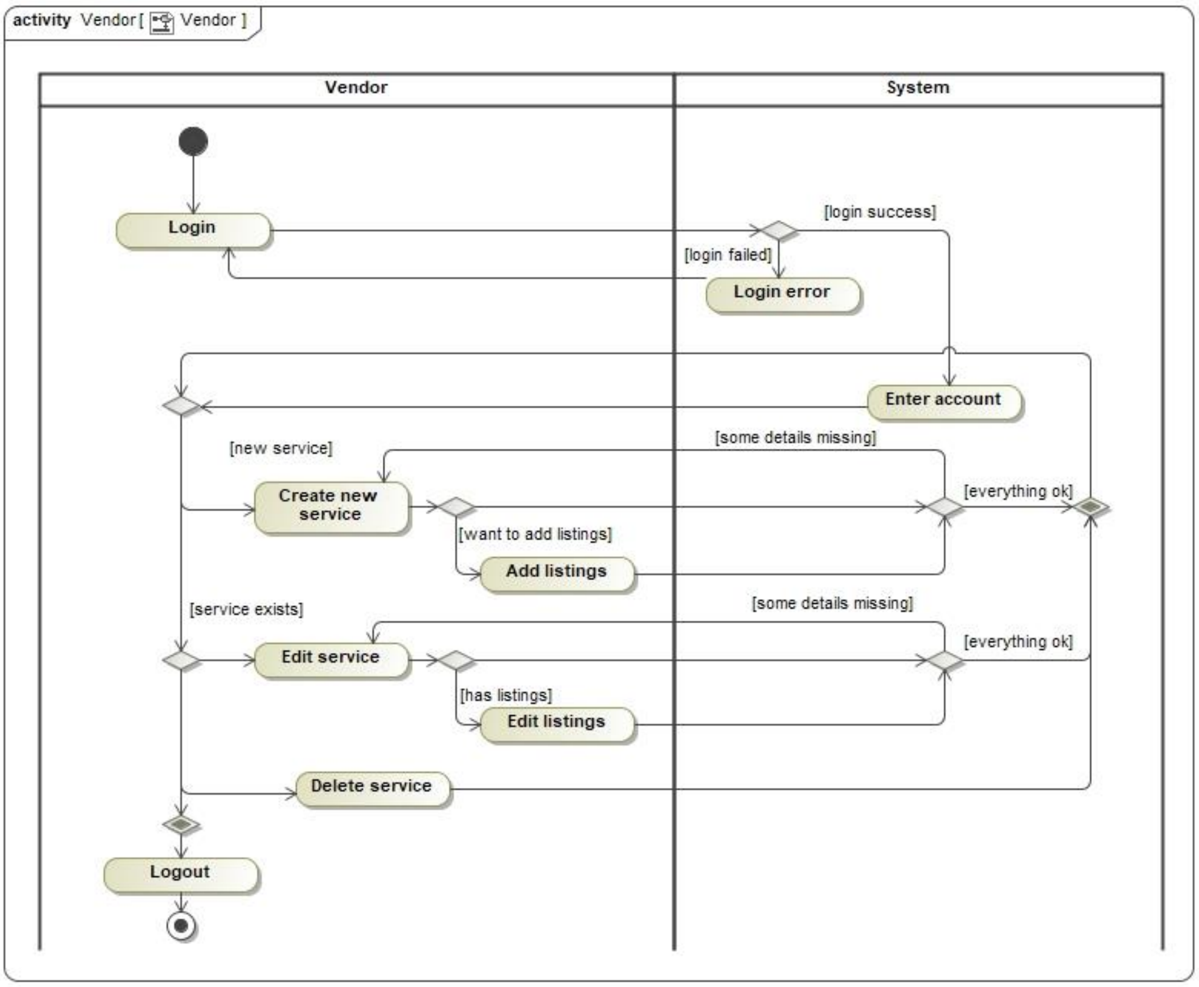

Figure 16. Vendor Activity Diagram After Changes

The main processes of buyer (customer) are covered in Figure 17 and Figure 18. The first diagram showcases that the buyer can currently search for services, and that after the changes the buyer will also be able to view services product listings, if they have any.

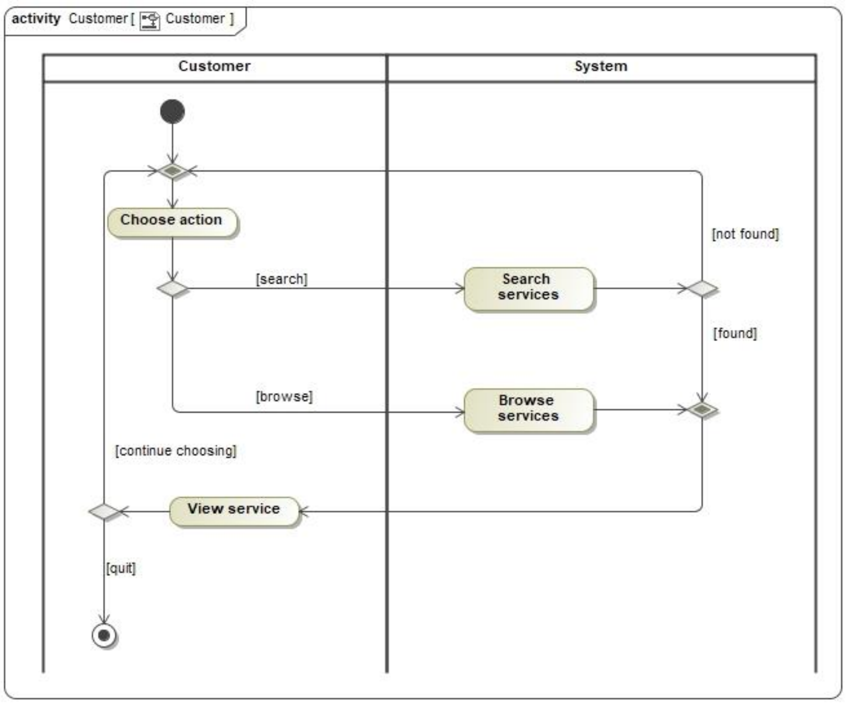

Figure 17. Buyer Activity Diagram Before Changes

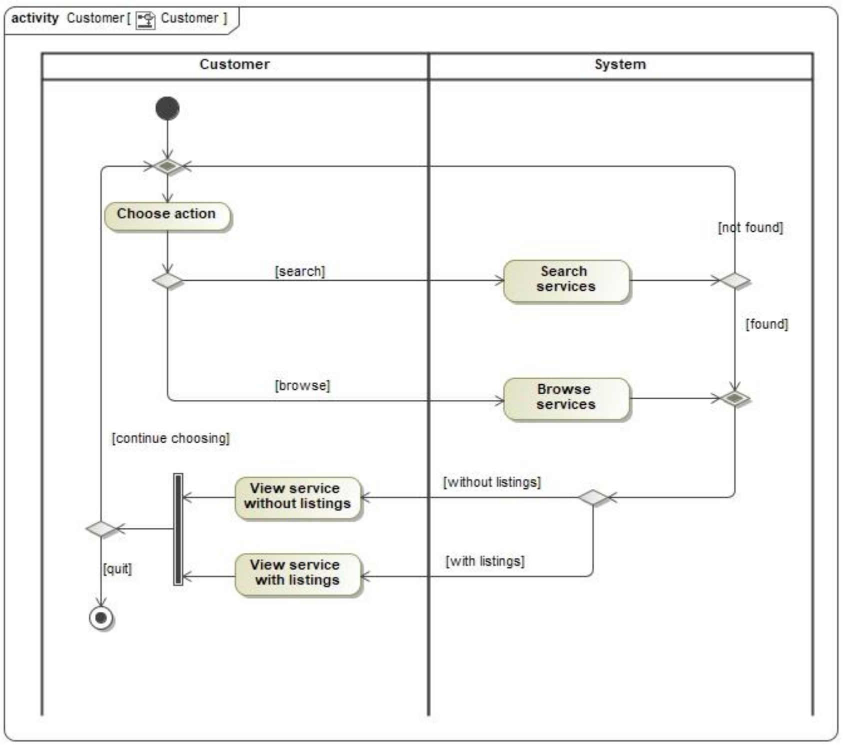

Figure 18. Buyer Activity Diagram After Changes

Figure 19 and Figure 20 illustrate the review process before and after the changes, respectively. After the changes, admin will be able to monitor reviews and services.

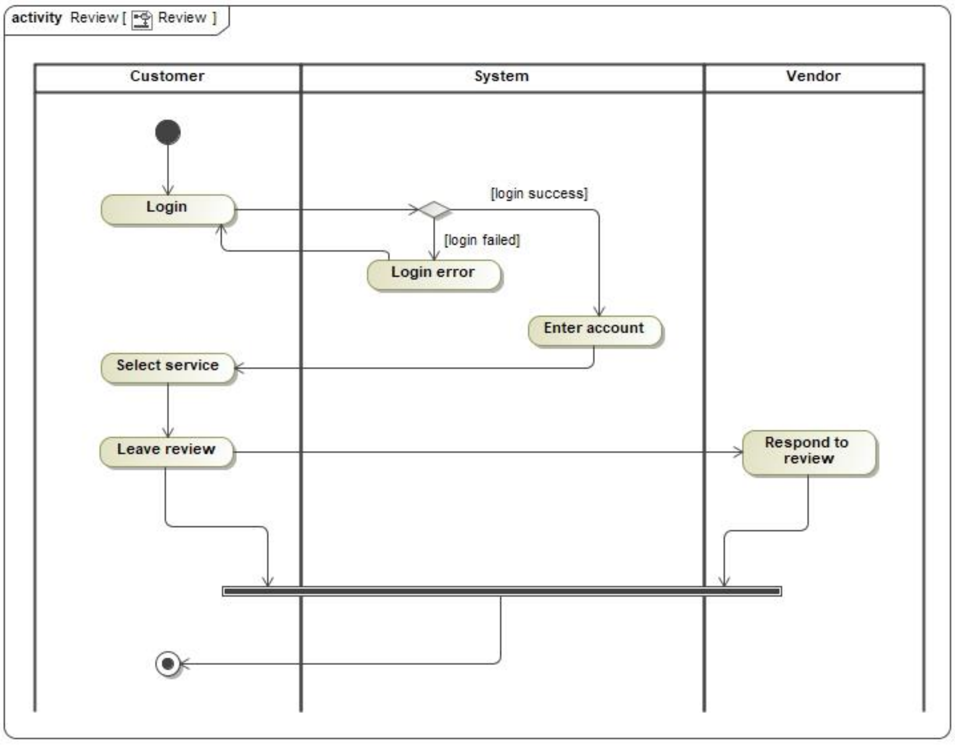

Figure 19. Review Activity Diagram Before Changes

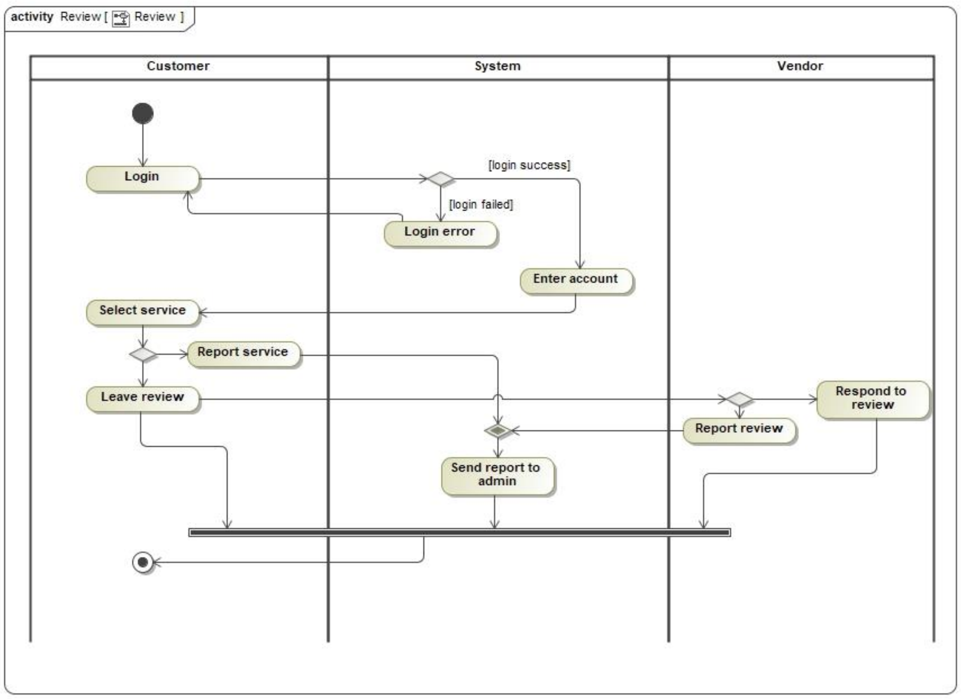

Figure 20. Review Activity Diagram After Changes

2.3.2 Sequence Diagram

In the sequence diagram on Figure 21 we can see the interactions between vendors (as users) and our system. Vendors first interact with the UI, which interacts with server-side controllers which interact with the database. In essence, all vendor (and other user types) activities interact in the same manner.

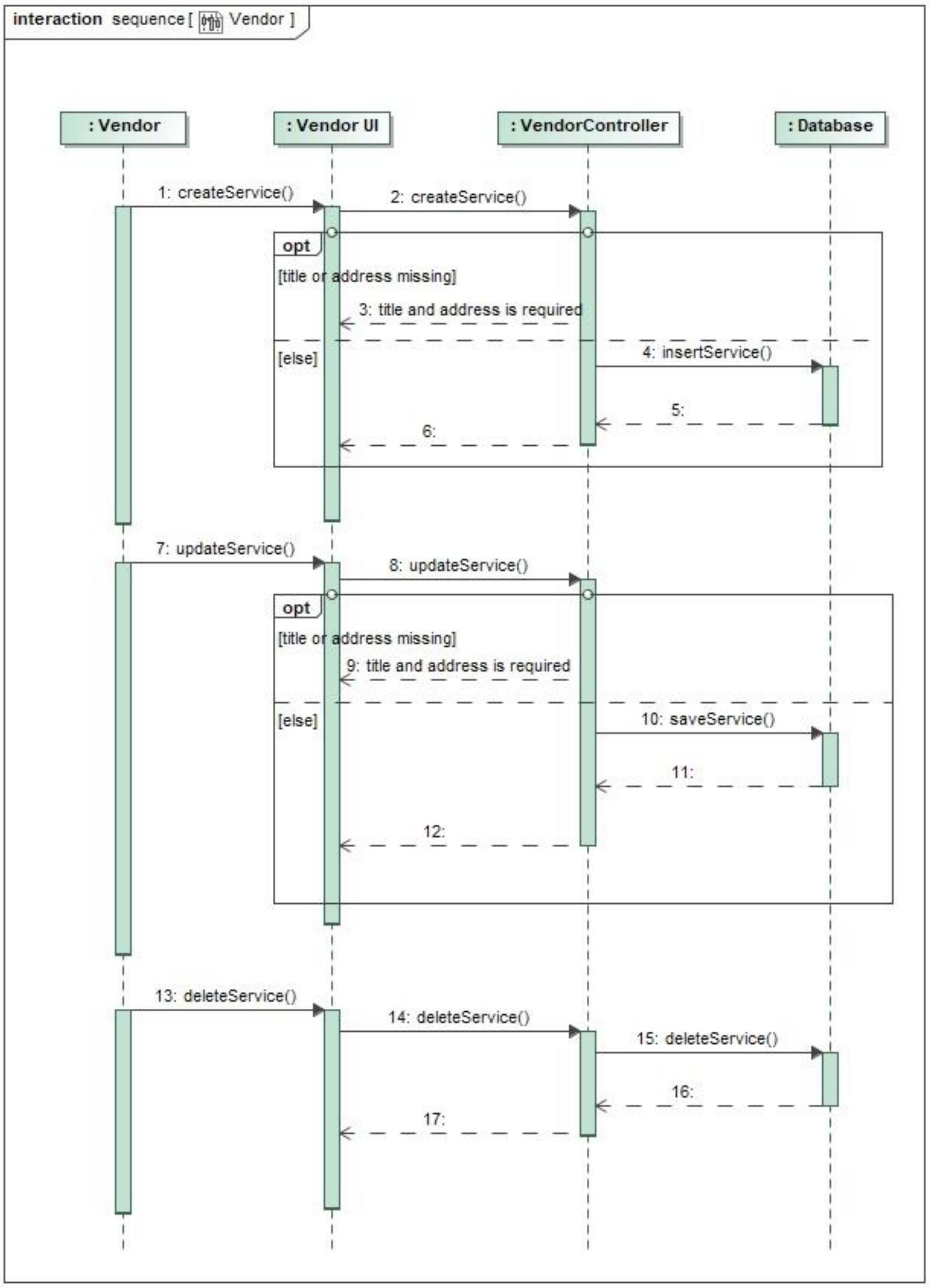

Figure 21. Vendor Sequence Diagram

2.4Physical View

In this part we analyzed the topology of software components on the physical layer as well as physical connections between these components. The system execution environments are displayed in Figure 22. User machine's operating system is not important since all types of users will access the system through the browser. Database does not necessarily have to be deployed on a Linux execution environment, but this is recommended. Execution environment of the server controllers is recommended to be a Windows operating system since it has been developed using the ASP.NET framework.

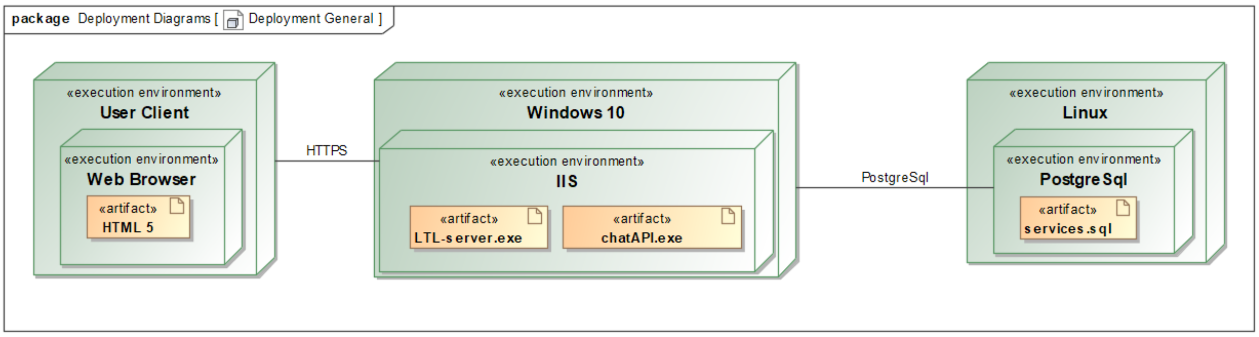
package Deployment Diagrams [Deployment General]
«execution environment» User Client
«execution environment» Web Browser
«artifact» HTML 5
HTTPS
«execution environment» Windows 10
«execution environment» IIS
«artifact» LTL-server.exe
«artifact» chatAPI.exe
PostgreSql
«execution environment» Linux
«execution environment» PostgreSql
«artifact» services.sql

Figure 22. General Deployment Diagram

Differences between development and production environments are shown in Figure 23 and Figure 24. The main difference in our opinion is that the API should be hosted on a separate server.

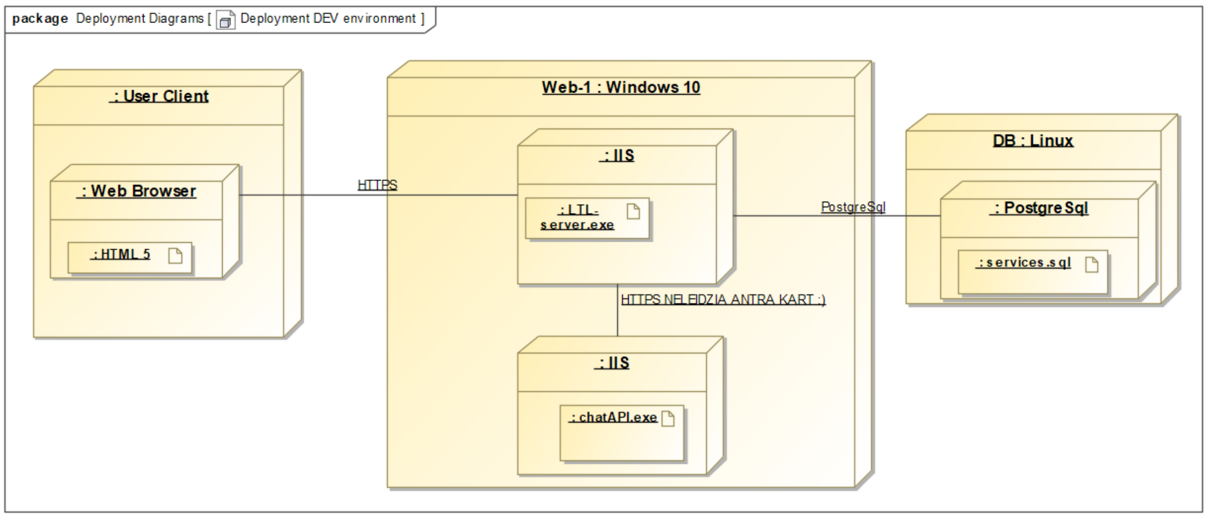

Figure 23. Development Environment Deployment Diagram

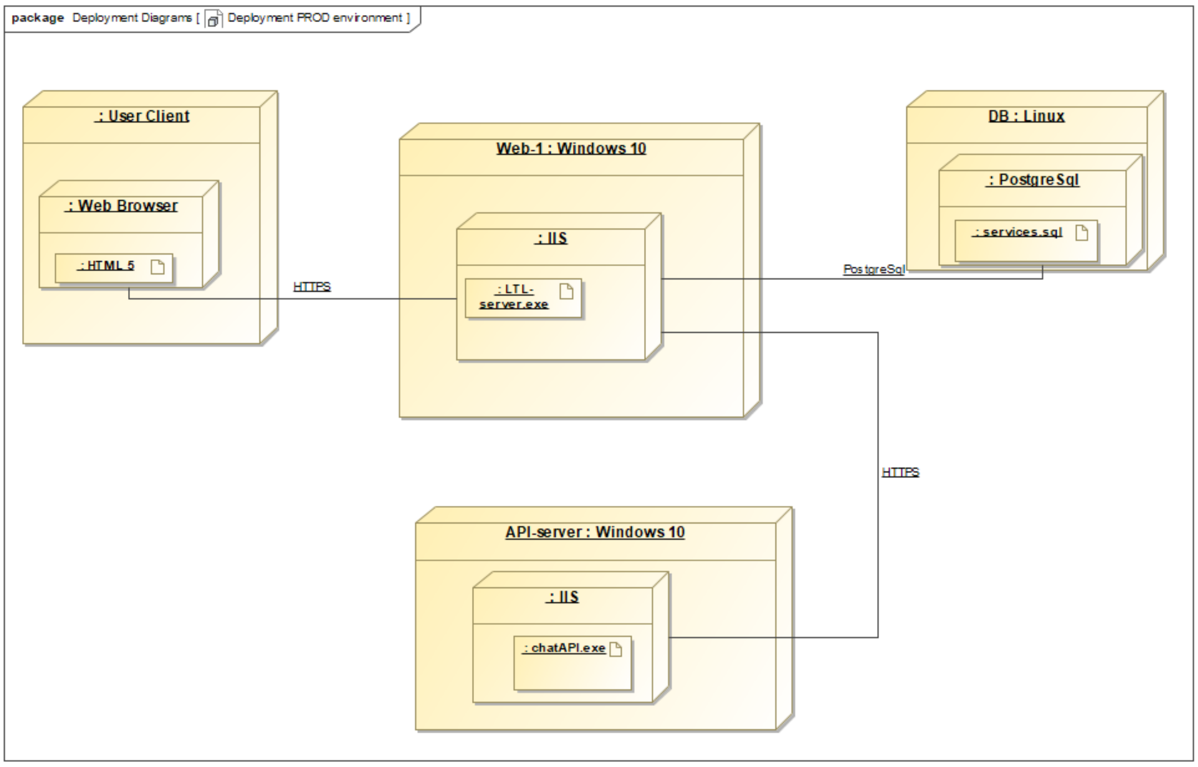

Figure 24. Production Environment Deployment Diagram

2.5Use Case View

This section will focus on the use case view of our architecture model. We have applied use case diagrams to illustrate sequences of interactions between objects, processes, and actors. With this information we can validify the architecture design and have a starting point for tests of an architecture prototype.

As our starting point, we have created a use case diagram overviewing the general use cases of our system. From this we have developed three more diagrams delving more deeply into the use case of each of the main actor groups.

2.5.1 Main Use Cases

The main use cases diagram shown in Figure 25, displays the main uses of our system by all groups of potential users, after the implementation of proposed changes. This diagram is a bit more complex than the ones to follow, however it is still a high-level overview. The main things to note is that the actual buying and selling of services is outside the scope of our system, even though it is the main goal of all users who use the platform. Another thing to pay attention to is that unregistered users also can use the platform, however they cannot leave any reviews or have other kinds of interaction with other users. All interactions will be possible to monitor by admins.

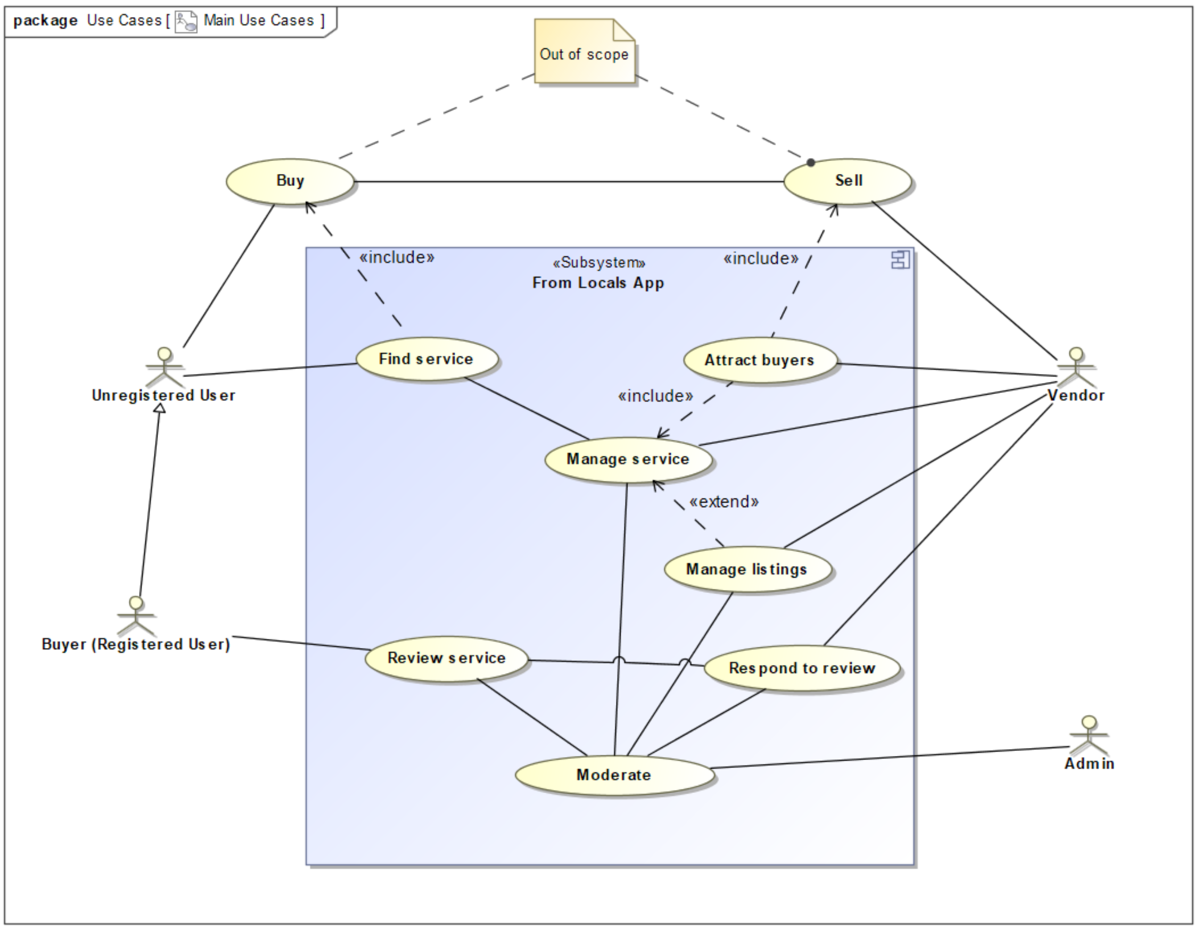

Figure 25. Main Use Case Diagram

2.5.2 Unregistered User Use Cases

Since unregistered users only have access to limited features, the corresponding unregistered user use case diagram, shown in Figure 26, is also straight-forward. The find service use case is shown to be a generalization of the two ways a user can find services and that both vendor accounts have access to the same basic features as buyer accounts.

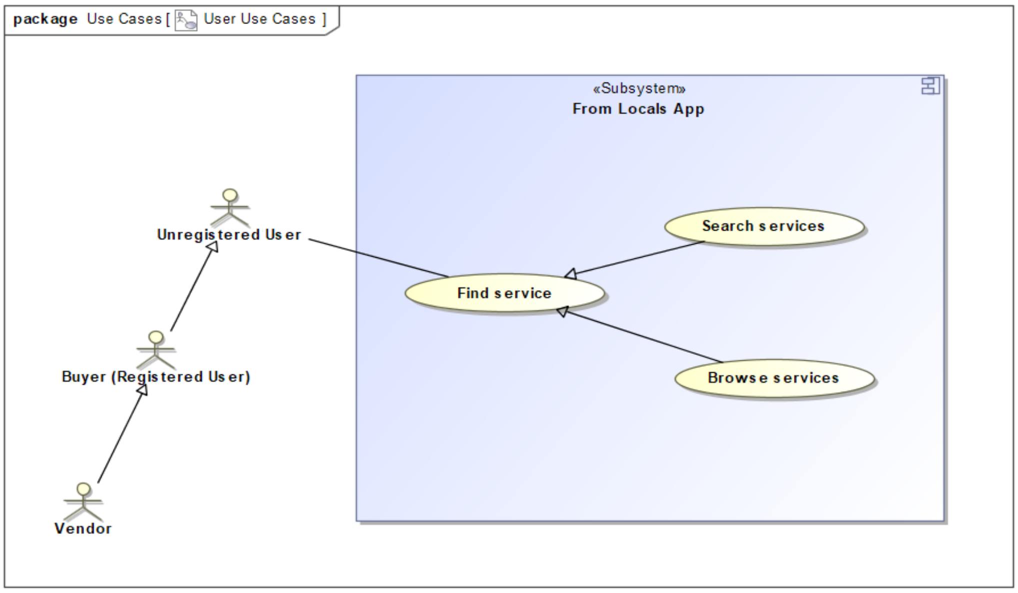

Figure 26. User Use Case Diagram

2.5.3 Buyer (Registered User) Use Cases

The buyer use case diagram, shown in Figure 27, displays that registered users have access to a lot more features than unregistered ones. However, authentication is required to access these features. Also, admin will be able to monitor activity by registered users and respond to their reports.

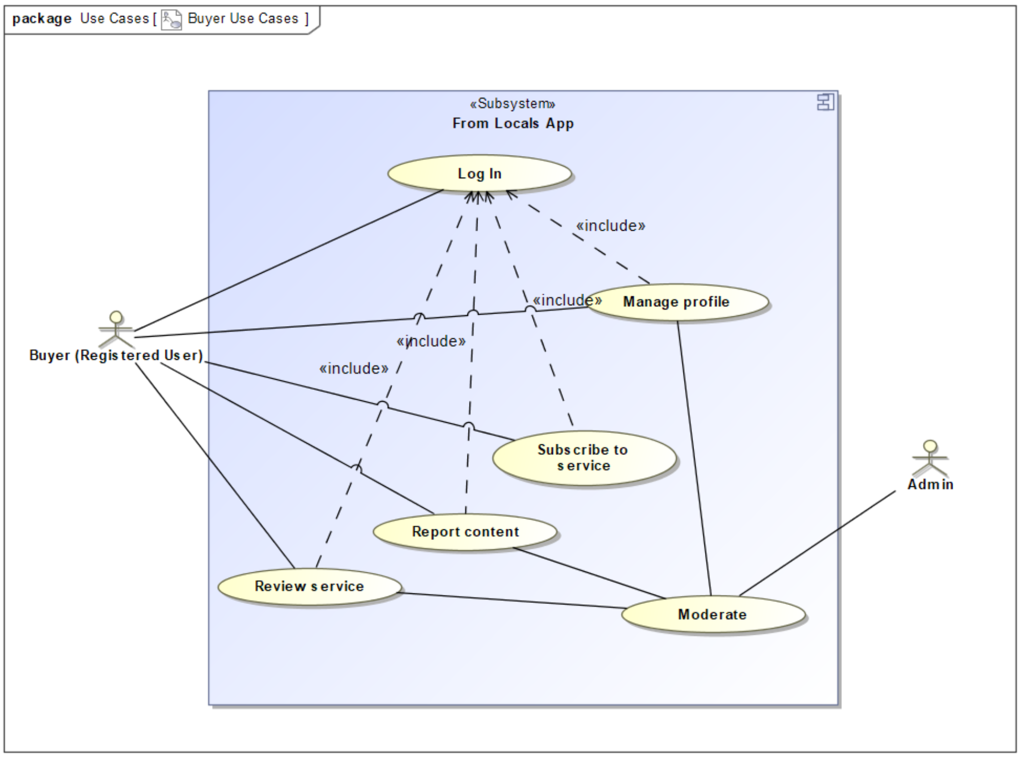

Figure 27. Buyer (Registered User) Use Case Diagram

2.5.4 Vendor Use Cases

The vendor use case diagram, shown in Figure 28, exhibits the possible uses of our system by vendors. Just like in the buyer use case diagram, vendors require authentication to have access to all the features of the system. As per our requirements, we will be implementing listings in services, allowing the option for vendors to add listings to their service. Again, admins will be able to moderate all the vendor activity.

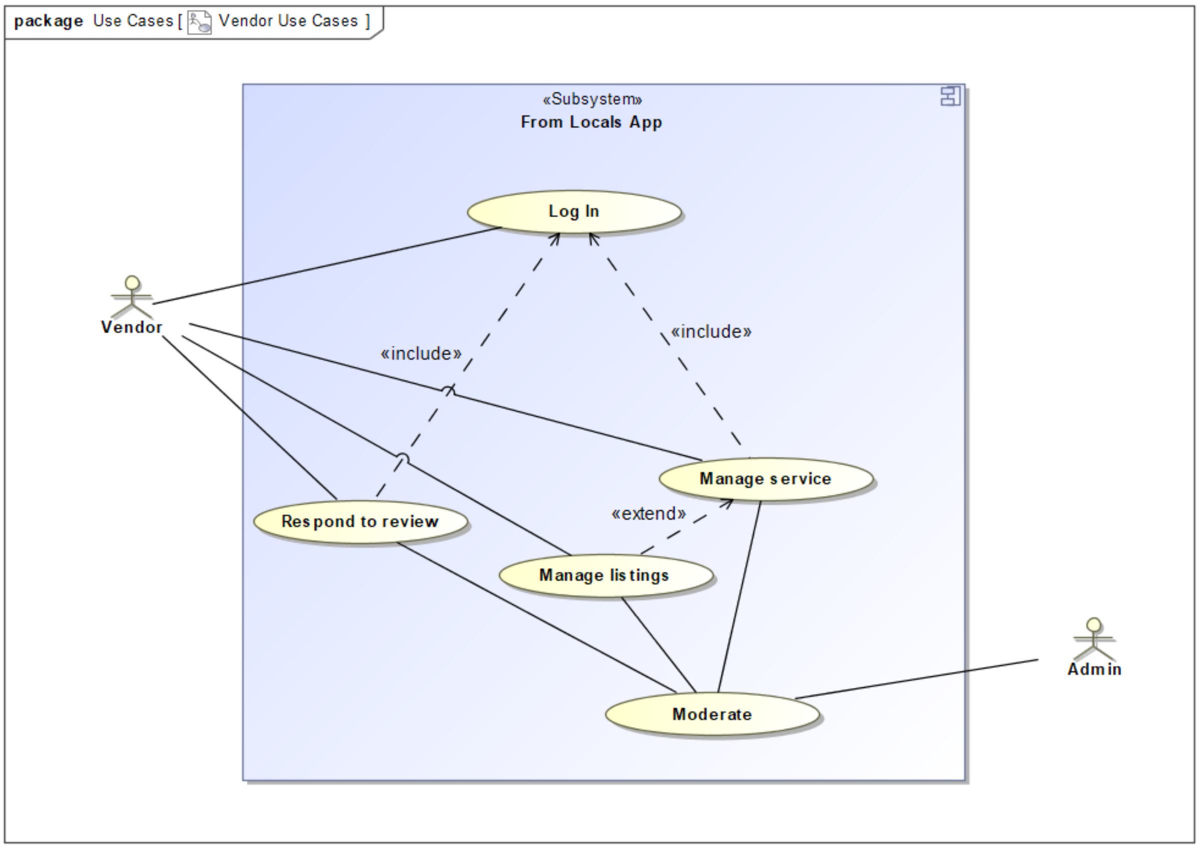

Figure 28. Vendor Use Case Diagram

2.6Model View to Requirements Traceability Matrix

We have created a view model to requirements traceability matrix shown in

Table 1 to help our team ensure all the requirements are covered in our planned architecture changes.

Table 1. Model View to Requirements Traceability Matrix

	Customer Requirement	R1. Every service to be able to add product listings	R2. Create admin and user roles	R2.1. Admins can delete any inappropriate content	R2.2. Registered users can report any inappropriate content.	R2.3. Create a separate admin page to handle reports.
Diagram/Model						
Activity	Customer					
	Review				X	
	Vendor	X				
Class	Before implementation					
	After implementation	X	X	X	X	
Component	General		X			X
	General decomp. now					
	General decomp. after		X	X	X	X
	Information flow					
Deployment	General					
	Dev. environment					
	Prod. environment					
Object	Object/General	X				
State Machine	User					
	Service current					
	Service future	X				
	Post Report			X		X
Use cases	Main	X	X	X	X	
	Buyer	X	X	X	X	
	User					
	Vendor	X	X	X	X	
Sequence	Vendor	X				
Package	General	X	X			

3. Conclusions

In our paper, we analyzed the current system state and modeled the system and upcoming changes using the 4+1 architecture view model. From the results of our work, we can draw several conclusions:

1. The current system is still in an early stage of development, with many features missing or not working properly. However, this does allow our team a degree of flexibility if needed to make major changes.
2. The tools and technologies the system is built with fit well with the purpose of the system, allowing easier development.
3. It will be challenging for our team to fix and add new features to a system that we were not involved in developing from the beginning.
4. The changes proposed, once implemented would benefit all user groups.
5. The new components to be implemented do not require major architecture changes and will fit in with the existing system.