G2U – a second life for every treasure

Course ID.: CPE-334

Submitted By-

Chawit Pimapansri	(ID: 65070503411)
Sorawit Tonpitak	(ID: 65070503438)
Jutamas Kaewchuenchai	(ID: 65070503444)
Nichaporn Manachaiprasert	(ID: 65070503446)
Thanakit Chokbunsuwan	(ID: 65070503448)
Arita Tragulmalee	(ID: 65070503470)
Yuil Tripathee	(ID: 65070503480)
Tom Medhi Pannier	(ID: 67540460025)

Submitted To-

Department of Computer Engineering
in partial fulfillment of the requirements
for the completion of
CPE-334 Software Engineering course.

Supervised by-

Dr. Natasha Dejdumrong
Associate Professor
Department of Computer Engineering

King Mongkut's University of Technology Thoburi (KMUTT)
Bangkok-10140, Thailand
1/2024

Revision History

Revision	Date	Author(s)	Description
v0.1	2023-04-21	Yuil Tripathee	First release, add outline
v0.2	2024-11-05	Tom Medhi Pan-nier	Add EU market evaluation

Abstract

We would like to think about it later.

Keywords: *We would like to think about it later.*

Terms, Acronyms, and Abbreviations

Keyword	Description
MVP	Minimum Viable Product

Contents

List of Tables

List of Figures

Part I

Project Description

<div align="right">

Chapter **1**
Introduction

</div>

1.1 Background and Motivation

As of 2024, we stand in the face of turbulent geopolitics, frequent extreme weather events, and escalating living cost all around the world. And we chose to make a small step today towards a greater impact tomorrow. Unreasonable consumer demands is believed to be one of the leading factors that is keeping us behind on our goal of developing sustainable societies and planet. UN SDG goal 12 promotes sustainable consumption and production patterns, ensuring efficient use of natural resources.

This inspired us to found G2U. We are developing on online e-commerce platform where every treasure has a find its second life, to a new owner. Finding the gap in this nascent niche market, we intend to incorporate lean startup model to gain market knowledge & feedback to the maximum. Then, our agile engineering team will capitalize on the market input swiftly to capture the sizable market share.

1.2 Market study

1.2.a SEA Market

1.2.b EU Market
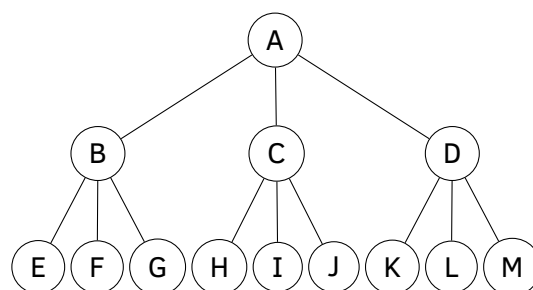
1.3 Scope of work

1.4 A dummy section

Figure 1.1: The proposed tree structure for the array implementation.

Chapter 2
Project Management

TODO: Comparative analysis of each methodology and what they deliver

2.1 Incremental funding methodology

Used for high level decisions.

TODO: Lean startup and cash model [1]

Instagram acquisition story on 2012 on $ 1 billion having 13 full time employees only. CPC from $ 0.5 to $ 3.5, CPM for $ 2 to $ 20 and SPE from $ 0.03 to $ 0.08. [2]

TODO: Project management (lecture 1) -> define scope, timeline, budget.

TODO: Communication and Rationale Management.

2.2 Agile Method with Kanban Tool

Used for low level decision and workflow orchestration.

Part II

Requirements

Requirements Elicitation

3.1   Elicitation Techniques

TODO: Before analyzing the system, various technique are employed to gather its requirements.
TODO: Explain:

- Interviews

- Questionnaires

- Workshops

- Observation

- Prototyping

3.2   Stakeholders

3.3   System Analysis - Data Flow

TODO: Data Flow diagram

3.4   Use Case Analysis

For software engineering observers, it is evident that use case diagram (in the Figure 3.1)
presented here is not in line with the data flow diagram. This is the result of agile practice. We
kept data flow analysis out of SDLC loop as it is redundant to sequence diagram (below). We
can effectively model both system behavior, reaction (in terms of data changes and transfers)
much effectively across our implementation team.

3.5   Functional Design

TODO: Here are some functional requirements (example)

- User Registration

- Tutor Scheduling and Availability

- Online class
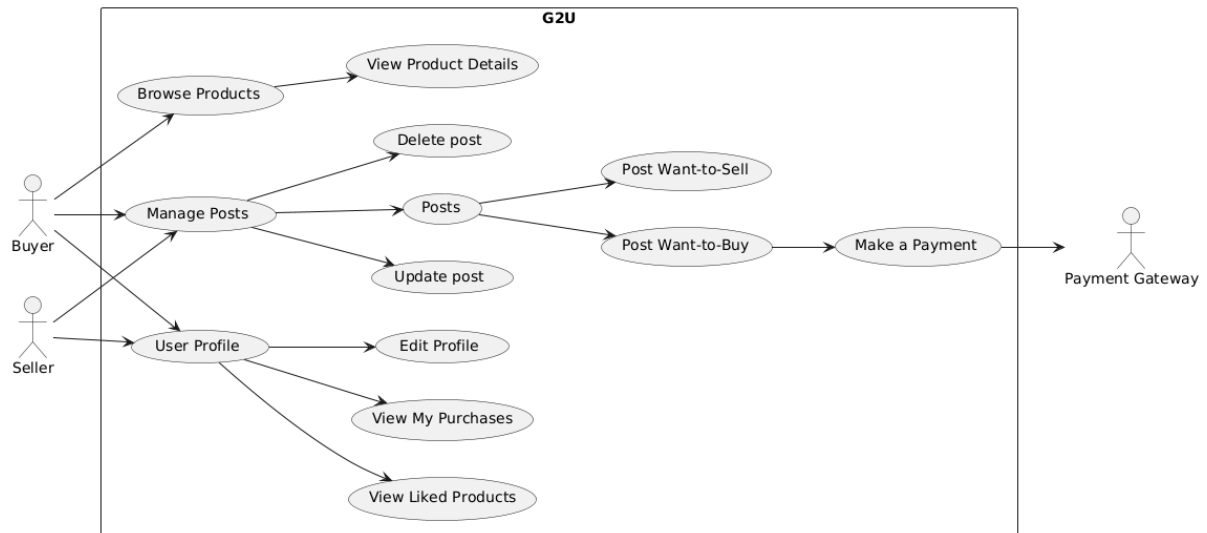
Figure 3.1: Latest version of use case diagram after feedback driven iteration

TODO: Translate this to user story when doing Kanban
TODO: Each functional requirements should have details and implementation in description list

3.6 Other Non-functional requirements

TODO: Quantize these requirements

- Scalability

- System Availability

- Security

- Usability

- Performance

3.6.a Mandated constraints

Examples include: Economics

3.6.b Regulatory compliance

Part III

Design and Development

Systems Analysis and Design

4.1 Software Analysis

4.1.a Class Diagram

The rational behind this implementation is to have fastest and the most iterable base for our back-end. It blends the OOAD concept to adapt with relational design that is readily deployed on PostgreSQL database.

4.1.b Sequence Diagram

The rational behind our sequence diagram is simple. We want to visualize and communicate how every machine components interacts in terms of interaction and data transfer in sequence of individual operations.
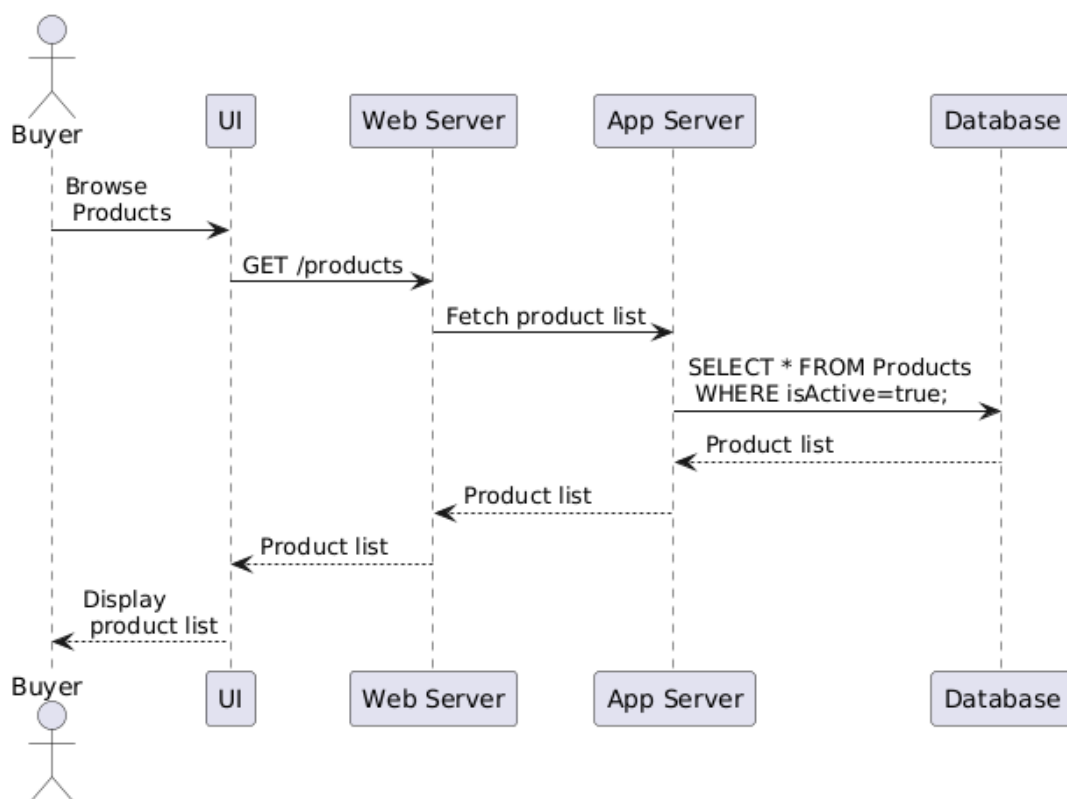

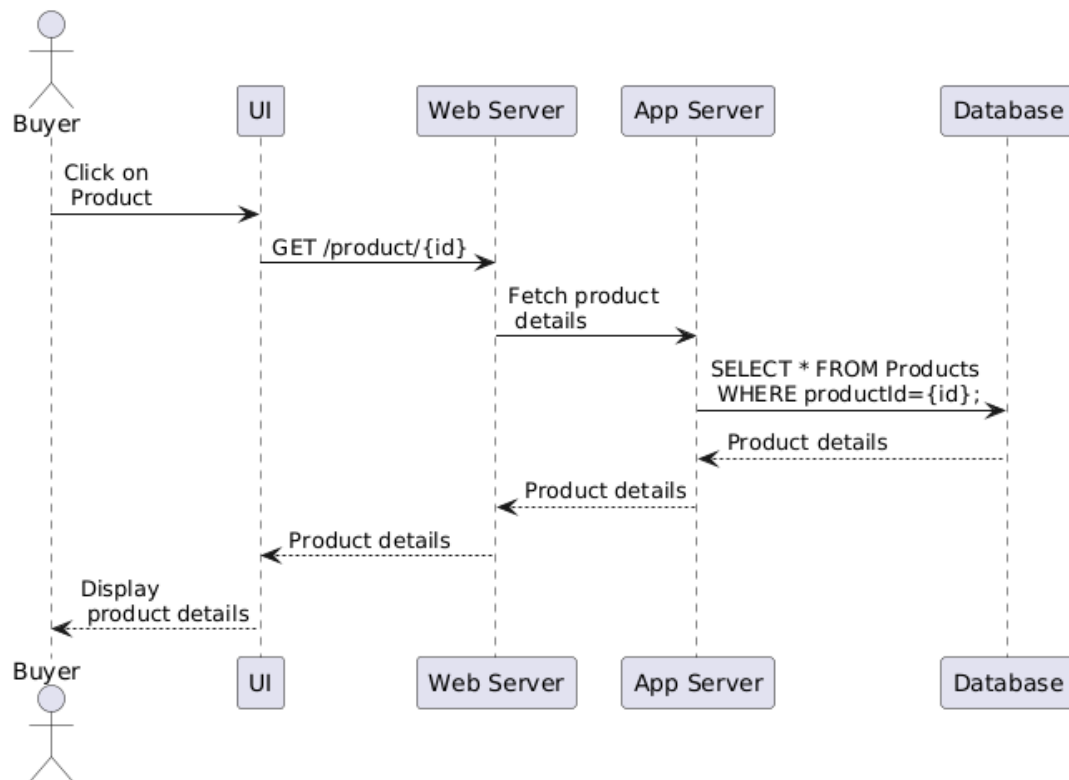Figure 4.1: Sequence diagram for Use-case: Browse Products

Figure 4.2: Sequence diagram for Use-case: View Products

4.1.b.1   Limitations

Unconventional implementation: Coding syntax such as including SQL as part of modeling app sequences. This is not part of best practice as specified in the UML specifications. However, it serves us more effectively as the means of communication for different technical sub-teams (front end, back-end and the deployment team). Our rationale behind this was to address the knowledge gap our current development team has. For the further updates, we intend to move up to the standard specification as knowledge gap is shortened and the back-end team is ready to receive requirements inputs using the top level description.

Less flexibility to stack changes For the current timeline, we chose the back-end stack to adapt customer needs as quickly as possible. However, if the requirements changes results in change of technical stack (for example, we switch from HTTP REST & SQL interface to GraphQL &gRPC); our specification for interaction sequences will be obsolete. The SQL–inspired specification cannot address changes to other modern day No-SQL schema (such as document store, column store). Therefore, if we change the database structure the specifics in the sequence diagram regarding SQL interface should be changed.
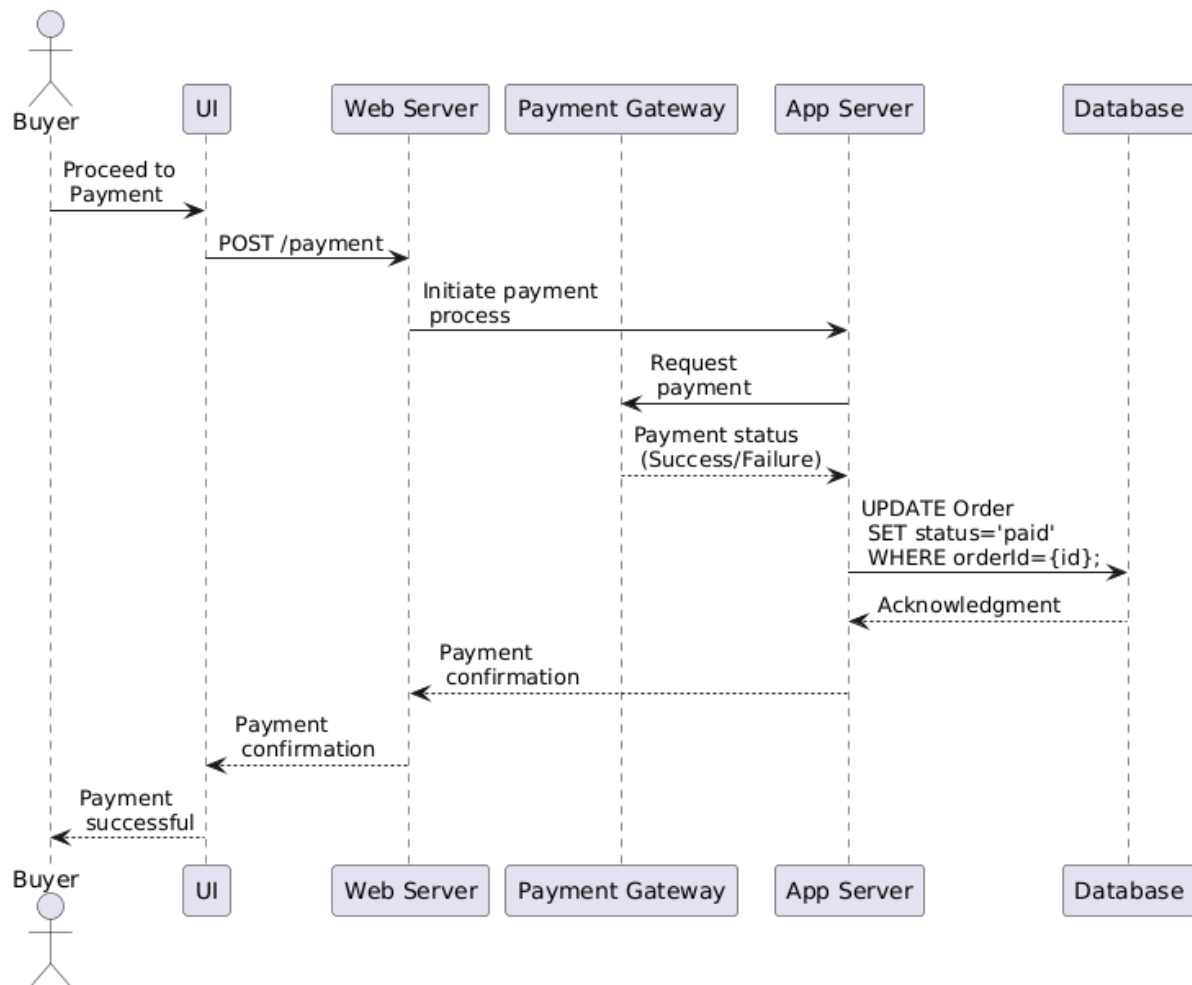
Figure 4.3: Sequence diagram for Use-case: Pay

4.2 Systems Design

4.2.a Demonstration model

Figure 4.7 is a minimal base on how our system is deployed and serving the user currently. We estimate to handle the peak of 10000 transactions per month comfortably in this setup. However, it might not meet the high up-time requirements as this server is currently hosted on our residential premises.

4.2.b Full scale production model

Upon reaching the designated transaction volume, we intend to migrate our application to a dedicated cloud service. Figure 4.8 suggests the tentative plan that allows us to deploy in cloud and scale.
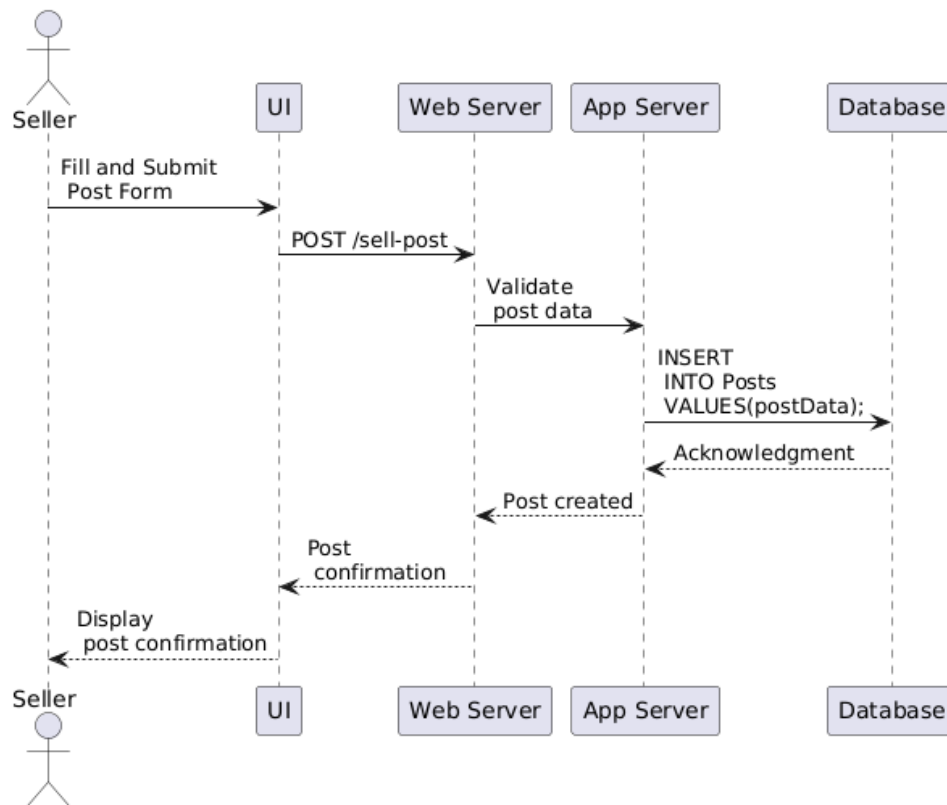
Figure 4.4: Sequence diagram for Use-case: Post new products
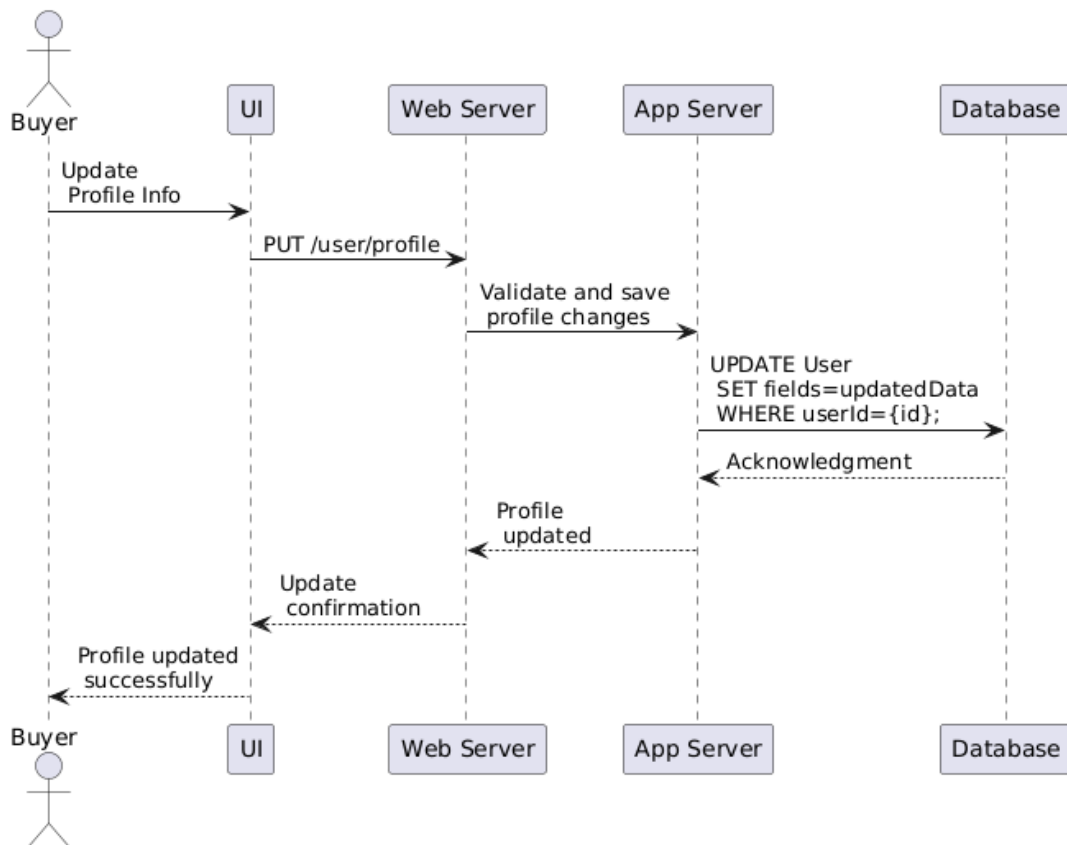
Figure 4.5: Sequence diagram for Use-case: Edit profile
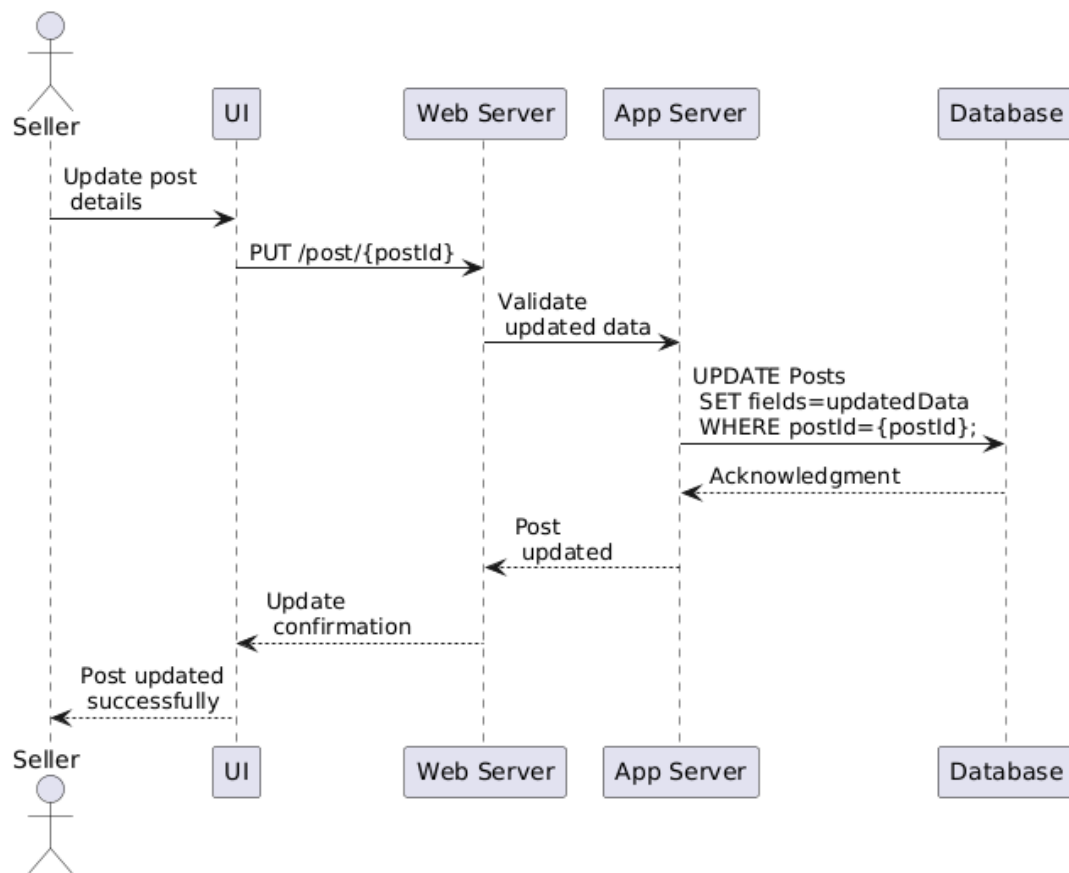
Figure 4.6: Sequence diagram for Use-case: Edit product post

Figure 4.7: Development phase components layout

12

Figure 4.8: Deployment phase components layout

<div align="right">

Chapter **5**

Implementation

</div>

5.1 **Low Code (Figma)**

5.2 **Prototyping**

5.3 **Coding**

5.4 **Systems Integration**

Part IV

Test and Evaluation

Evaluation of Outcomes

6.1   Testing Methodologies

So far, testing code correctness has been least of the issue. There we will continue performing higher level tests to confirm user's perception of our app. Here are the few methods we found appropriate:
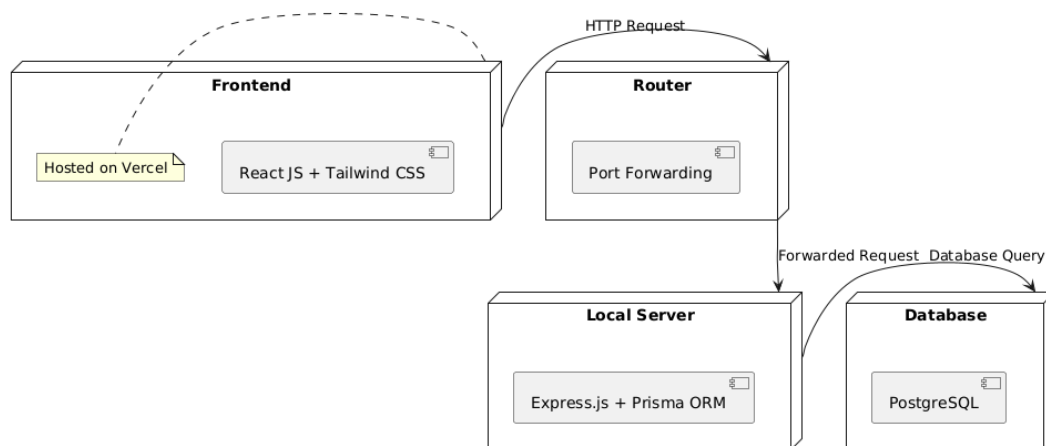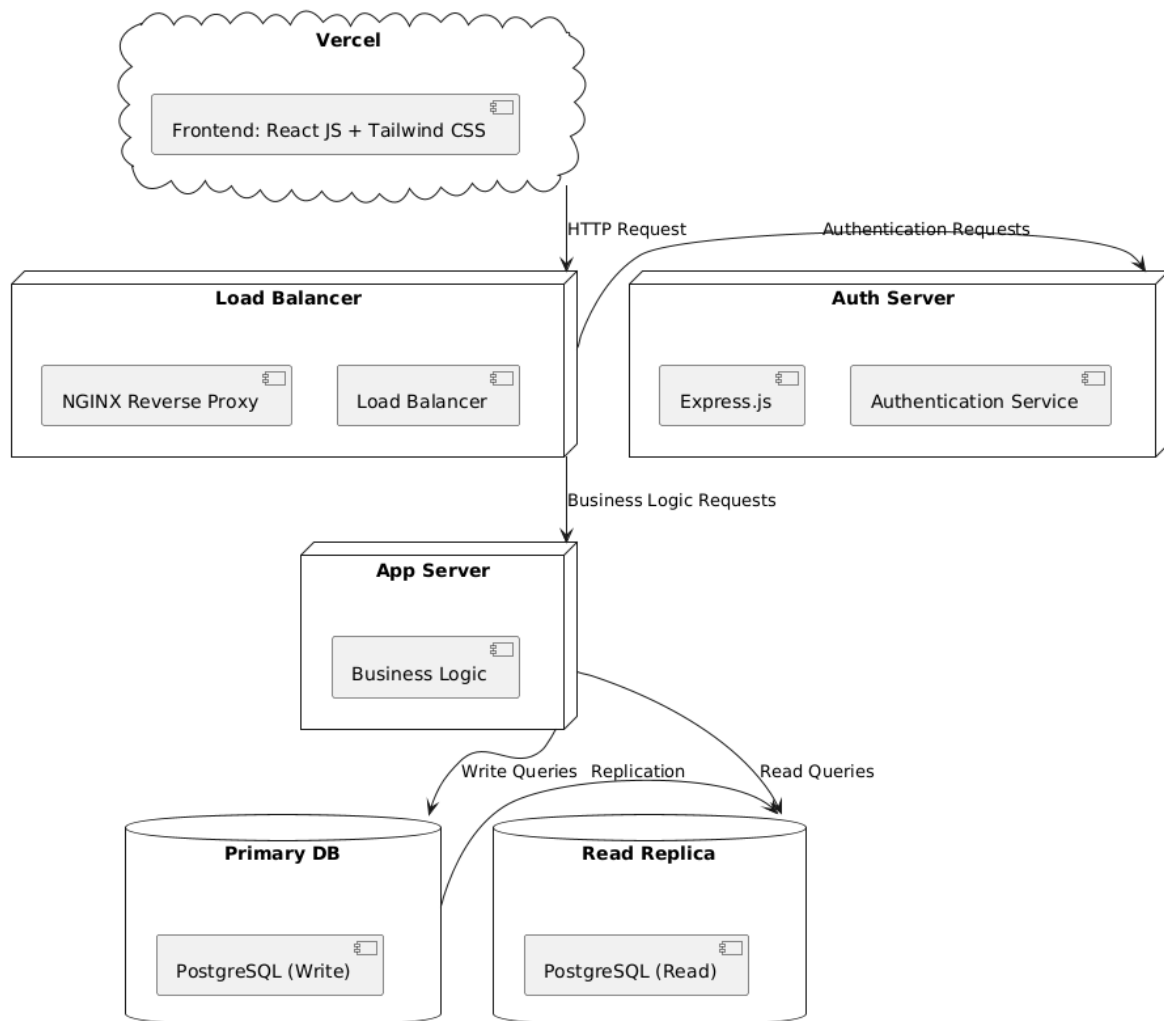
A/B Testing We consider this as the most important test to retain user's feedback. If we give only one interface for user to test, users perceive this as an open ended questionnaire. It makes the participants in the test anxious and fall back to their previous experiences with the similar platform. But, if we provide the tester with two set of alternatives, it is quite easier to make comparative evaluation. Click up rate test compared in different applications helps us to exploit the things users are not willing to semantically express in the test participation (verbally or in written).

Integration Testing Such tests allow testing multiple parts of the software system as the group. We perform this test to see if two modules are interfacing well together, as this is an internal test. This test sets an important precedence before we bring our platform for review with the external users.

Stress Testing This is on the technical side and least relevant for now. Since our user base is small, this test has no use so far till now. But, in the coming future we might have to go through vertical and horizontal integration. We have to know how well the app can handle the load and where optimizations need to be made.

6.2   Results

We have observed A/B testing being tried and tested in by multiple tech companies. Facebook and YouTube gradually roll out their test features and set it as optional. Now, only thing left is to wait and watch user's behavior, especially which service do the users want to stay longer on. The classic example being Apple rolling their flagship MacBook Pro lineup with touch-bar and phasing it out after facing criticism.

We were able to replicate such methodologies my letting users make comparative evaluation between our previous MVP and iterative successor version of our platform. This test also blends with agile method which allows improvements in our platform with the least overhead. We have discovered this method to be the most effective in our test experience and it provides significantly better actionable information that lets the improvement of our application.

<div align="right">

Chapter **7**
Conclusion

</div>

7.1 Discussion

Aligning to the lean startup model, we allowed ourselves to effectively adopt agile engineering workflow. We set the correct success criteria to obtain customer feedback as much as possible, rather than following our prejudice of how an e-commerce platform should be.

With the flexible team setup and feedback driven development, we gladly announce on this date, that we have achieved to develop and deploy an MVP (minimum viable product), that is good enough for closed testing within trusted circles.

7.2 Future Work

In the days to come, we intend to keep improvising our platform based on user feedback from various domains. In summary, we will rewards users to educate us on what they want and then deliver exactly that.

In summary, our technical road map for FY 2024/25 is presented as below. The goal of the technical team is to reward the participants (as much as possible) to retain and maintain the customer base.

Q4 2024 Iterate the existing version of our application to adhere with security and business standards for European market.

Q1 2025 Improvise the existing version of our application to get market readiness to launch in Thai market.

Q2 2025 Integrate AI language model to improve localization and User Experience.

As for our business road-map, our goal is to sustain customer development process and keep getting credible feedback. As long as we have credible feedback, we are in the upper hand of business.

Q1 2025 participate in marketing festivals, book event launches, and university open house as supporting sponsors.

Q2 2025 experiment and launch giveaway platform and include influencers to participate (those with good follower base on Instagram and Tiktok).

Q3 2025 launch lottery quest to maintain customer retention.

7.3   Recommendation

7.3.a   Opportunities

We are grateful to achieve these key distinctive learning outcomes, which is hard to achieve otherwise.

Secrets to quality software We learned (in a somehow hard way) that better code can make program better, but not a better software. We consider this to be the most important learning outcome. Until the software serves the intended need of user in the most effective way possible, software engineering processes should undergo necessary revisions by capturing user feedback with as much as actionable information as possible.

Software Engineering Economics A software engineering projects requires positive cash flow to survive and sustain. We explored and discovered several ways the project can be sustained in a long run with positive cash flow. This is crucial to us because we're going to explore the solution in a blue ocean environment. In such cases, the rules to deliver business values are not yet established, neither standardized. This requirement is quite relaxed in some environment though (like the companies who receive certain fees to complete outsourcing contracts). For us, optimum way to sustain our cash flow were scarce. However, lean startup came into rescue. It allows us to create a testing ground, while giving us space to test and validate new business models with optimum resource utilization.

Synergy When we practice certain discipline for long time in quest of being a subject matter expert in a specialized domain (say server side development), it is quite easy to get tempted and fall into trap of being a possible single point of failure. This is a very risky move because in a short run, we might observe a fast progress. In a long run, lack of presence of that stakeholder alone can jeopardize the engineering operation. Therefore, we found a healthy practice to have each individual team members get a slight generalized knowhow of their few peers. If need be, any potential crisis due to the absence of that developer can be mitigated before the project faces an existential risk.

7.3.b   Challenges

If we had to do it over again, we learned that following challenges should be solved before the implementation phase begins:

Line of communication We have when programming team members are unfamiliar to each other due to some reasons (such as unmatched learning curve). Front end and back end teams have different backgrounds that shape their idea of integrating two ends together. There are some workable solutions to try out such as add a personnel specializing in full stack development. Role of such person would be to develop interfaces in the back end part and sharing data directly to the front-end team as they like. In this case both parties can maintain consistency when there is changes to schema or business logic. However, this luxury might not be available all the times. Therefore, a common documentation such

as shared OpenAPI specification [4] is found to be better alternatives. In order follow this approach, either team (front end or back end) has to put their desired specification in a shared document. Every time there is a breaking changes from either team, it has to go through this document first prior to code changes.

Retaining quality user's feedback This is perpetually existing challenge in teams of all tiers (from startups to established enterprises). User testing is easier said than done. Because of the competitive market, developer teams have dilemma in terms of trade offs for confidentiality before their app release versus information they should publicly release. Businesses generally can not follow academic style survey methodologies due to various regulatory and technical reasons. This led us to conclude that we always need to explore ways to understand and track user appeal to our features.

References

[1] Steve Blank. Everything about lean startup in 12 minutes, July 2024. URL https://www.youtube.com/watch?v=G-wwOK4X0lc.

[2] Investor Relations at Meta (Previously Facebook). Press release - facebook to acquire instagram, April 2012. URL https://investor.fb.com/investor-news/press-release-details/2012/Facebook-to-Acquire-Instagram/default.aspx.

[3] Duolingo. Duolingo brand guidelines, December 2024. URL https://design.duolingo.com/. Accessed on 2024-12-04.

[4] swagger.io. Openapi specification version 3.0, December 2024. URL https://swagger.io/specification/.