An Automated Negotiation System for eCommerce Store Owners to Enable Flexible Product Pricing

Jake O'Halloran

A dissertation submitted in partial fulfilment of the requirements of Dublin Institute of Technology for the degree of M.Sc. in Computing (Advanced Software Development)

January 2019

I certify that this dissertation which I now submit for examination for the award of MSc in Computing (Advanced Software Development), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the test of my work.

This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and has not been submitted in whole or part for an award in any other Institute or University.

The work reported on in this dissertation conforms to the principles and requirements of the Institute's guidelines for ethics in research.

Signed: __Jake O'Halloran____________________

Date: **4th January 2019**

ABSTRACT

If a store owner wishes to sell a product online, they traditionally have two options for deciding on a price. They can sell the product at a fixed price like the products sold on sites like Amazon, or they can put the product in an auction and let demand from customers drive the final sales price like the products sold on sites like eBay. Both options have their pros and cons. An alternative option for deciding on a final sales price for the product is to enable negotiation on the product. With this, there is a dynamic nature to the price; each customer can negotiate with the store owner on the price which allows the final sales price to both change over time and on a customer by customer basis. The issue with enabling negotiation in the context of eCommerce is the time investment needed from the store owner. A store owner cannot negotiate every time an offer comes in from a potential customer, the potential time investment would not be acceptable. Using software agents to automate the process of negotiation for the seller is a potential solution to enabling negotiation in eCommerce for store owners. In this research, a system such as the one just described is developed in a way that mirrors real life negotiations more closely and after evaluation, is found to be a potential solution for the enabling of negotiation in eCommerce.

Key words: eCommerce, automated negotiation, software agents

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my supervisor Damian Gordon, whose guidance contributed enormously to the production of this dissertation.

I would like to thank my Mam and Dad for their support and making this possible.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 CHAPTER 1 INTRODUCTION

1.1 Background

Online marketplaces are websites that facilitate products trading between Internet buyers and Internet sellers (Ngai, 2007). Currently, the two most popular methods for trading products through these marketplaces are fixed price selling (e.g. Amazon) and online auctions (e.g. eBay). Despite their popularity, both methods have drawbacks for both sellers and their potential buyers. Fixed price selling lacks flexibility; a price must be set from the get-go and unless later updated, is the only price available to buyers. This requires sellers needing to decide on prices very carefully. Too high and buyers will be lost, too low and the sellers profit margin will fall. Online auctions fix this by allowing sellers to set a reasonably low starting price and letting demand drive price even higher. This reduces the fear sellers can have that setting a certain price point will drive away potential sales. However, if bidding in the auction is low, this can yield sales prices that are lower than if sellers would have used a fixed price. This is where a third method comes in, negotiation. Negotiation is a form of decision-making where two or more parties jointly explore possible solutions in order to reach a consensus (Rahwan, 2002).

Sample negotiation: Alice is selling a product for €120.00 - the negotiation issue is the Product Price.

1. Bob offers to pay a price of €50.00
2. Alice counters Bobs offer with €105.00
3. Bob offers to pay a price of €75.00
4. Alice counters Bobs offer with €95.00
5. Bob offers to pay a price of €82.00
6. Alice counters Bobs offer with a final counter offer of €85.00
7. Bob and Alice agree on €85.00

Negotiation has become very interesting research issue (Fathey, 2005). A clear advantage when buying in a real world setting rather than online is the ability to negotiate. When buyers and sellers are face to face, negotiation can take place for a more mutually agreeable price. The potential advantages of which is the buyer gets a price they are more comfortable paying and the seller gets the opportunity to sell a product they might not of had the opportunity to sell if the price was fixed. Negotiation is becoming an emerging area in the evolution of electronic commerce (ecommerce) on the web. Having a system that can facilitate negotiations between sellers and buyers in the context of an e-commerce site would be very beneficial to both parties, as it would be a way to integrate the advantages of negotiation in real world marketplaces into online marketplaces.

1.2 Research Question

Research into negotiation systems that serve to aid both buyers and sellers has been well documented, but research into systems designed only for sellers is lacking. Furthermore, research into a negotiation system designed to specifically aid eCommerce store owners in negotiating with potential customers is currently absent. When a customer enters a store and desires to negotiate for a lower price on a specific product, store owners may have had previous dealings with that specific customer. In real world negotiations, store owners can use information gathered from past negotiations to get better deals in future negotiations. For example, if the customers average goal within negotiations is known to the store owner e.g. the customer always agrees on any price that is 20% below the asking price or better, the store owner can use this information to both ensure they come to an agreement with the customer and sell the product to them for the highest price they can. As different customers negotiate in different ways, the negotiation strategy employed by the store owner when negotiating with one customer may be different when negotiating with another. Negotiation systems that use past negotiation results (Amgoud, 2005), customer data (Kulkami, 2017) and multiple negotiation strategies (Kexing, 2011) have already been researched. However, these have not been used in correlation with each other, not in a way that attempts to mimic real world negotiations more closely and not with the aim of trying to benefit ecommerce store owners exclusively.

A huge issue with enabling negotiation for these store owners is the time investment needed from them. A store owner would have to take part in a negotiation each time an offer for a product is sent by a potential buyer. This process would create significant issues if store owners were to try to keep up with the influx of product offers (potentially thousands per day). As humans could not possibly do this work efficiently, software agents could be used as a way of taking care of this work in place of a store owner. Agents are software components that act on behalf of users, or group of users (Fathey, 2005). The use of agents to support Ecommerce operations, especially in automating the buying and selling process is promising (Yu, 2008).

The project will explore automated negotiation and will focus on application of the research to the specific setting of eCommerce. A negotiation system will be created to enable negotiation functionality for ecommerce store owners. The negotiation system will allow customers to negotiate on the price of products, and the system will negotiate with the customers directly, in place of the store owner. The system will negotiate in a way that mimics real world negotiations more closely than other existing negotiation systems. Using a dynamic negotiation strategy influenced by a combination of negotiation and customer data attained by examining past negotiations with each customer individually, the system will attempt to agree on a price with customers that benefits the store owners.

1.3 Research Objectives

The research objectives of this project are as follows:

- Review existing literature to get an understanding on the current knowledge and sentiments held in the field of automated negotiation. Use this to aid in the design and development of a new and alternative system.
- Design and develop a basic negotiation system. This system will be capable of automating negotiations with customers for ecommerce store owners. This system will use a simple, static strategy when negotiating with customers.
- Improve on the basic negotiation system, enabling it to utilize a dynamic strategy through analysing and evaluating past negotiations with individual customers so to achieve better results in its future negotiations with those same customers.
- Prepare an experiment to allow evaluation of the negotiation systems performance.
- Prepare a second experiment to evaluate the usability of the negation system.
- Evaluate the results of the experiments and the interview and provide analysis.

To accomplish the above objectives, the research question and hypothesis are defined as follows:

- Null Hypothesis (H0): A negotiation system used in an ecommerce setting cannot use a negotiation strategy that mimics real life negotiations to provide better results for a store owner.
- Alternate Hypothesis (H1): If a negotiation system can use a negotiation strategy that mimics real world negotiations to agree on prices with customers that are to the satisfaction of a store owner, then the system is an acceptable tool to enable negotiation in ecommerce.
- Alternate Hypothesis (H2): If customers have positive experiences negotiating with the automated negotiations system, then the system is an acceptable tool to enable negotiation in ecommerce.

"Can a negotiation system implemented in the context of an eCommerce site, use a combination of customer and past negotiation data to negotiate pricing with customers to the satisfaction of the owner of the website?"

1.4 Research Methods

The project will start with secondary research of the existing literature to gain a better understanding of both the topic of negotiation in eCommerce and the existing state of the art approaches to providing solutions for it. The secondary research will be focused on areas that could help with the research problem and included: Negotiations in eCommerce, automated negotiation systems and non-automated negotiation systems.
Primary research will follow which will be a combination of both qualitative and quantitative methods. Combining both methods allows for the analysis to cover both the effectiveness of the system and its usability. Interviews with volunteers who agree to negotiate with the system allows for the capture of data that could be used to measure the usability of the negotiation system. Focusing on its ability to successfully integrate with an eCommerce site. Quantitative methods will be used to compare the results attained by the basic negotiation system with those attained by the improved negotiation system.

1.5 Project Scope

The scope of the research will be to design, develop and evaluate a negotiation system that uses past negotiation data and customer negotiation patterns in its negotiations with customers in relation to a negotiation system that does not use the data in its negotiations. The proposed negotiation system will only ever be tasked with negotiating on a singular issue: product price. When a customer negotiates with the proposed system, the system has only one goal: agree to sell the product to the customer for the highest price it can for owners of ecommerce stores. The system will not be concerned with coming to agreements on prices that benefit the customers, or agreements that are mutually beneficial. The system will aim to end negotiations with prices that most benefit the store owners.

1.6 Limitations

The first limitation of the system will be its inability to deal with customers with dynamic negotiation patterns. The negotiation system will operate under the assumption that the max percentage of the product price a customer is willing to pay during one negotiation will be the same for any follow up negotiations. Once the system learns a customer's behaviour once, it will not relearn it going forward.

A limitation when it comes to evaluating the system will be the fact that it will be necessary to create a large dataset of negotiation results from scratch. The system will be built from the ground up and so a dataset of negotiation results will have to be created for the new system before evaluation can begin. An existing dataset cannot be used due to the individual nature of the project.

To evaluate the systems performance accurately, a great number of negotiations must be completed. The number of completed negotiations needed will be much larger than what could effectively be achieved by using real life volunteers to negotiate with the system. As such, mock customers will have to be created and made negotiate with the system in place of real-life customers. This will be a limitation on the research as the level of variety on the types of customers that can be created will be limited. In the end, only 16 customers with different negotiating behaviour will be created, which cannot completely capture the amount of potential variety in the negotiating behaviour between real life customers.

With a system that will be biased towards achieving results that only satisfy the store owner, a potential limitation is whether customers will be able to detect this. If they can, this could have some potentially negative results. Customers may not engage in negotiations with a system they know to possess a bias against them.

1.7 Thesis Outline

Chapter 2 outlines the existing research on negotiation systems to form a foundation from which to begin the design and development on the proposed negotiation system and reveal ways to evaluate the system once it has been implemented.

Chapter 3 describes the development process that will be used for the project and details the technology that will be used for the development before giving detailed designs for the negotiation system and its architecture.

Chapter 4 explains how the negotiation system was implemented by breaking it down feature by feature and providing detailed explanations and code samples. The two experiments that were going to be used to evaluate the implemented negotiation system were described.

Chapter 5 evaluates the results gathered from undertaking both the experiments. Through this, the system is described from the perspective of its abilities and limitations.

Chapter 6 presents a conclusion to the thesis. The key points from each chapter are gathered together and discussed to give an overview of the findings and their contributions to the body of research into negotiation systems as they exist in the context of the internet.

2 CHAPTER 2. NEGOTIATION: AUTOMATED, NON-AUTOMATED, AND SATISFACTION

2.1 Introduction

In this chapter, multiple areas of research are studied to form a foundation from which to begin the design and development on the proposed negotiation system. Firstly, background information is given on certain topics to lay the groundwork for a better understanding on the existing negotiation system research. Once a basic understanding has been established, existing literature on previous approaches taken to creating negotiation systems are described, examined and evaluated. Reviewing the existing literature will reveal and explore two main approaches to providing solutions for negotiation systems; automated and non-automated. Lastly, the suitability and importance of using satisfaction with negotiation results as an evaluation measure for negotiations is explained and justified.

2.2 eCommerce

The internet has revolutionised how commerce is performed (Pease, 2001). The Internet is a nearly perfect market because information is instantaneous, and buyers can compare the offerings of sellers worldwide (Srinivasan, 2002). The onset of this "nearly perfect market" came with it a new form of commerce; electronic commerce. There is no universal accepted definition of electronic commerce (Ngai, 2002). Electronic commerce (eCommerce) has been defined in several ways depending on the context and research objective of the author (Grandon, 2004). From an online perspective, eCommerce provides the capability of buying and selling products and information on the Internet and other online services (Ngai, 2002). It is undeniable that eCommerce has changed many things in the business; it not only has changed the way they sell, purchase or deal with their customers and suppliers but it has also changed the business perspective from "production excellence" to "customer intimacy" (Rahayu, 2015). eCommerce is an exciting area for research, because of its relative novelty and exploding growth (Ngai, 2002).

2.3 Software Agents In Negotiation

When looking at literature that describes the approaches to designing negotiation systems, it becomes quite evident just how popular the use of software agents are as integral parts of the solutions. The jobs they are tasked with completing in each system differ on a system to system basis. In some systems such as the ones designed by Kersten *et al*. (2003) and Chen *et al*. (2005), the agents aid users in the negotiation by assessing and critiquing offers. Other systems such as the ones developed by Wang *et al*. (2006) and Cheng *et al*. (2006) use agents to negotiate on behalf of the users. Rahwan *et al*. (2002) used a combination of the two, not only would agents be used to negotiate on behalf of the users, other agents would also be given the responsibility of assessing and critiquing the offers made by these negotiation agents.

When research on twelve existing negotiation systems was examined, all but one of the designs were found to use software agents either to aid the negotiations in their systems or to undertake the negotiations themselves. In all twelve of the systems, multiple agents were used and in all but one, agents were used to either represent or aid both the buyer and the seller. The set up for the agents varied among the twelve approaches, with many to many, one to many and one to one all being present among the research.

2.4 Negotiation: Basics

Negotiation is a process between buyers and sellers in a business transaction trying to reach an agreement on one or more issues (Junyan, 2007). In ecommerce, these issues traditionally include price, quantity or delivery. The negotiation process is an iterative back and forth of messages containing offers, counter offers, final offers and ends with either side rejecting or accepting an offer sent to them on a certain issue based on their goal for that issue.

Presented in the figure below is an example of a negotiation on the issue of price on a product worth 100 euros. The buyer has a goal to agree on a price of 85 euros or below. The seller has a goal to agree on a price of 80 euros or above. After a third offer was sent to the seller, they once again countered it, but with the specification that it was the final offer, as any lower would result in them not achieving their goal. At this point, the buyer must either accept or reject the price of 80 euros. The negotiation ends as the buyer accepts the final offer of 80, as their goal was 85 euros or below.

Message Number	Buyer Message	Seller Message
1	Offer: 60 euros	
2		Counteroffer: 90 euros
3	Offer: 65 euros	
4		Counteroffer: 87 euros
5	Offer: 70 euros	
6		Final Offer: 80 euros
7	Accept Offer	

Figure 1 Sample Negotiation 1

Presented in the figure below is another example of a negotiation between a buyer and seller on the issue of price on a product worth 100 euros. The buyer has a goal to agree on a price of 80 euros or below. The seller has a goal to agree on a price of 85 euros or above. Here the buyer sent a third offer to the seller, but with the specification that it was the final offer, as any higher would result in them not achieving their goal. At this point, the seller must either accept or reject the price of 80 euros. The negotiation ends as the seller rejects the final offer of 80, as their goal was 85 euros or above.

Message Number	Buyer Message	Seller Message
1	Offer: 60 euros	
2		Counteroffer: 90 euros
3	Offer: 75 euros	
4		Counteroffer: 87 euros
5	Final Offer: 80 euros	
6		Reject Offer

Figure 2 Sample Negotiation 2

2.5 Existing Approaches

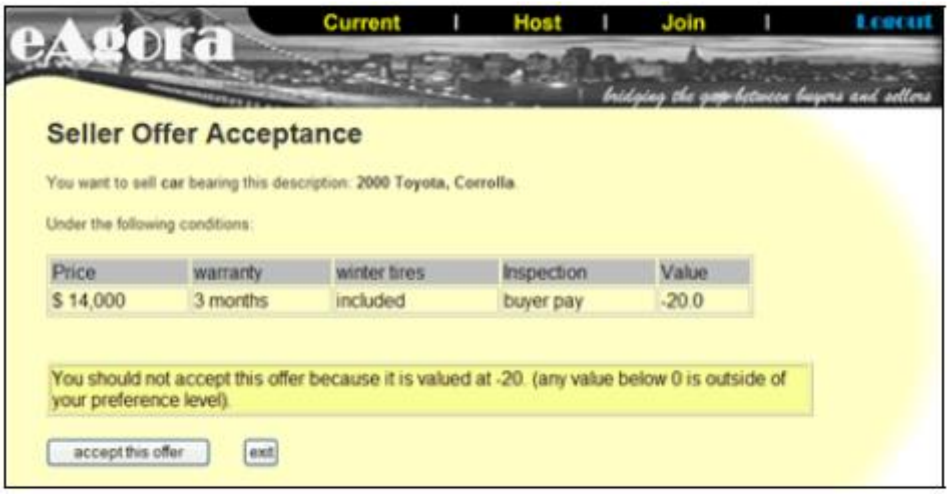
In this section two distinct approaches to negotiation will be reviewed, first non-automated approaches, and then automated ones.

2.5.1 Non-automated Approaches

Non-automated negotiation systems are used to simplify, aid or facilitate the process of negotiation for users.

Kersten *et al.* (2003) developed a web-based negotiation environment designed to support negotiation via software agents and negotiation support/decision systems. The agents' purpose in the system was to interpret the negotiation activities for the user and provide methodical advice based on them using game theory analysis. Agents offered assessment of each offer that was received during negotiations, along with a quantitative evaluation. After receiving feedback on the systems prototype from users, Kersten *et al.* concluded that negotiation software agents specific to negotiation are useful features when supporting internet-based negotiations.

In a paper by Chen *et al.* (2005) an e-marketplace named eAgora was discussed. EAgora enabled buyers and sellers to engage in negotiations about a multitude of issues. Software agents generated and critiqued offers. The agents did not negotiate themselves, they would generate several different offers from which the human negotiator could pick from, hence aiding in the negotiation process. In this system fuzzy logic was used in the analysis of offer generation. Usability tests were held to gauge the usefulness of the agents. Participants were asked to use eAgora with agents help and without. Post experiment questionnaires revealed that 92% of users were in favour of using a system like eAgora to sell their products online. The tests also revealed a correlation in the number of successful negotiations when the agents were used versus when not, supporting Kersten *et al.* conclusion about agents being useful for supporting negotiations. Users of the system did note a desire for the system to utilize a multi-strategy negotiation protocol instead of just the 4 that were with the system during the test.



Seller Offer Acceptance

You want to sell car bearing this description: 2000 Toyota, Corrolla

Under the following conditions:

Price	warranty	winter tires	Inspection	Value
$ 14,000	3 months	included	buyer pay	-20.0

You should not accept this offer because it is valued at -20. (any value below 0 is outside of your preference level).

Figure 3 eAgora system critiquing offer made to user

In a paper by Kulkami *et al*. (2017) it is stated that getting the best price for a product is what prospective buyers are focused on. In response, the paper proposed a system that e-negotiates by observing potential buyers' surfing patterns. These patterns included sites visited and products surfed. The system provided custom discounts to potential buyers based on these patterns. The paper argued that providing a custom discounted price motivates buyers to make purchases. Kulkami *et al*. explained that this exploits the human psychology which craves for attention, by making the user feel special. That when a user whose patterns indicate they are interested in a product see a discount unique to themselves, they will be tempted to make the purchase.

The sentiment held by Kulkami *et al*. about price being the focus for prospective buyers is supported by a paper by Ngai *et al*. (2002) where it is argued that conflicts related to the price of a product is usually the bottleneck in trading efficiently within online marketplace. In response, an embedded negotiation support system was proposed to allow buyers and sellers to negotiate with each other about product price through system interfaces, supposedly leading to greater efficiency within the trade. The system brought with it the issue of demanding lengthy user engagement time. This was fixed by introducing a semi-automated negotiation process. Regression test results proved that the introduction of semi-automation did not damage the original systems negotiation success rate, making the proposed system a potential solution to the bottleneck of product price deliberation.

In a paper by Kumar *et al*. (2005), middleware to support online bilateral negotiation was implemented. The middleware was designed to be integrated with trading systems such as auctions, helping with the bidding and bargaining processes. To support various users of the system, multiple negotiation protocols were available to be utilized to achieve differing objectives the various users might have (find best deal, find fastest deal etc). The system used a pre-negotiation stage to enable negotiation of the rules of the negotiation. In addition, a platform was provided to allow users to add or delete issues of the negotiation (price, quantity etc), allowing for dynamic changes to be made to the process of the negotiation. This type of system has the advantage of supporting multi issue, where others described above seem fixated on the negotiation issue of price.

2.5.2 Automated Approaches

In automated approaches to negotiation, software agents are used in place of and to the benefit of the users within the negotiations.

Karp *et al*. (2004) took a unique approach when presenting a strategy for automated negotiation. In this strategy, the same approach is taken as computer programs that play chess; a game tree is built. For every offer, every counteroffer is looked at, and every counteroffer to each of them, and so on. The strategy then selects the counteroffer with the larger expected payoff. Like the 2003 paper by Kersten *et al*., this paper showed that a game theory approach is a possible approach to negotiation but through experimentation, found it doesn’t provide any great advantage to alternative strategies. In fact, many problems arise from using the game theory strategy. For example, a ‘bad move’ e.g. ‘bad’ selecting a larger than ideal counter offer, could potentially penalize both the buyer and the seller, as this could result in the negotiation ending without an agreement being met.

Offer and counteroffer selection described above has also been studied in a paper by Amgoud *et al*. (2005). This paper studied strategy for selecting which offers get made by agents during negotiation. A general setting was proposed for defining a strategy, with its parameters being what is argued as what should be the three mental states of an agent: its beliefs, its goals and its rejections. This paper argues that most work in the sector of offer selection only considers goals of the negotiation and instead rejections should also play a key role in offer selection.

In a study by Cheng *et al*. (2006), a heuristic model was presented for making trade-offs in automated negotiation. A fuzzy inference system was used to allow agents to tactically make trade-offs within a third-party e-marketplace. The proposed automated negotiation system was evaluated using five different experiments. The results of the experiments found the fuzzy system to be efficient in terms of the number of offers exchanged. The paper also noted that while in their system each negotiation is treated independently of the others, they think it could be advantageous if the agents could apply information from one negotiation to other negotiations. This information could include what was suggested above by Amgoud (2005), where it was suggested that negotiation rejections could play a role in offer selection.

Experiment	Description	Results
1	Negotiation success rate with different groups	Average success rate: 94.31
2	% of negotiations that reached optimal agreement	Average: 93.86
3	Issue weighting effect on negotiation results	Number of Offers increases
4	Effects of increasing number of issues	Only effects were increased offers
5	Quantitative vs Qualitative issues negotiated on	System produced better results on purely qualitative issues

Table 1 Summary of 5 experiments ran by Cheng (2006)

Wang *et al*. (2006) presented a fuzzy logic based autonomous negotiation agent that was designed to reach mutual agreements efficiently and effectively by interacting with other such agents over various sets of issues on behalf of real-world parties. It is argued in this paper that conventual methods such as game theory that are utilized in systems such as the ones described above by Kersten *et al*. (2003) and Karp *et al*. (2004) are incapable of being used in automated negotiation in open environments like the internet where information is full of uncertainty, whereas fuzzy approaches are quite suitable. When evaluating the negotiations made by the agent, the researchers used the MATLAB fuzzy logic toolbox and concluded that their proposed offer evaluation techniques are acceptable and implementable.

More *et al*. (2002) proposed a negotiation system for use between a provider and consumer that utilized the cloud. The cloud stored product and agent details for the system. In the system, both provider and consumer have agents that handle their side of the negotiation so that the negotiation process is fully autonomous. Data on the user and their product requirements are used by the agents to negotiate on features or issues, with the agents provided ultimately notifying the users on the negotiation success or failure. They found the cloud to be beneficial to negotiation in ecommerce due to its inherent security and scalability which is needed for the data being stored and being able to handle the dynamic number of agents that would be needed for each consumer.

Rahwan *et al*. (2002) presented a framework for enabling one to many negotiations via concurrent one to one agent negotiation. The system used a duel layer strategy by using individual and coordination agents. Agents negotiated on behalf of parties in a cyclic fashion. After each cycle, another agent would evaluate how each agent was doing and issued new instructions to them. This system was tested in an ecommerce environment with a personal computer trading scenario. Results from the test showed that one to many ("seller to buyers") negotiation is possible by implementing a system that coordinated parallel one to one ("seller to buyer") negotiations. The proposed framework solves the problem of scalability that was brought up above by More (2002) that is needed for ecommerce negotiation.

In a paper by Huang *et al*. (2002), a model is presented for automated negotiation through agents on the internet. In this model, the process of negotiation is driven by the internal beliefs of the agents involved. Agents can change their behaviour (and results) by adjusting some critical parameters in an easy and flexible way. The different 'personalities' can be shown be easily plugging in subjective beliefs to the agent. The paper showed that having different 'internal beliefs' can result in agents behaving differently. Some humans are harder negotiators than others, and so it is argued that agents need dynamically adapt based on who they are representing. Through simulations, the strength of having internal belief updating for agents was demonstrated.

The idea of a multi-strategy negotiation system discussed by Chen (2005) was also researched by Cao *et al*. (2015). They developed a prototype negotiation system based around agents and multi-strategy. In this system, the negotiation agent selected the appropriate strategy dynamically to deal with the ever-changing offers in order to get a successful agreement. To evaluate the protype, buyer and seller agents negotiated on a single price issue, with only the buyer agent having the strategy section ability and the seller agent according to a fixed negotiation strategy. Experiment results confirmed that multi-strategy negotiation selection leads to higher counterpart acceptance ratio. The researchers argue that strategy selection models should be requisite components in negotiating agent architectures, as their research demonstrates significant potential in enhancing the efficiency of negotiations in ecommerce. This supports the need for dynamic strategy among agents suggested by Huang *et al*.

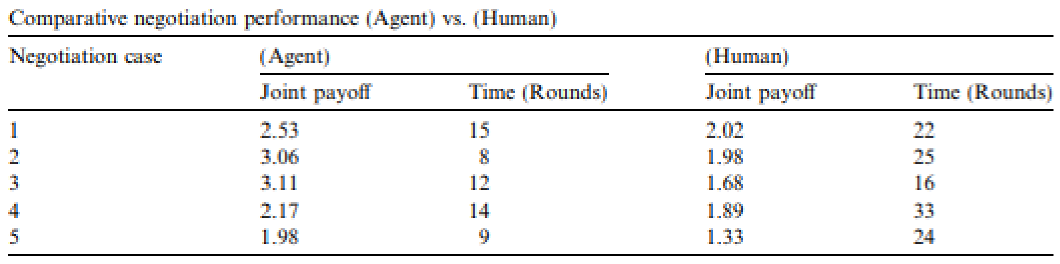
This is also mirrored in a paper by Junyan *et al*. (2007) that stated it is important for negotiation agents to choose a proper strategy that is based on negotiation factors to achieve its maximum utility with their negotiations. In this paper a negotiation model was presented that was based on four of these factors:

- belief,
- time,
- competition, and
- opportunity.

Furthermore, an adaptive negotiation strategy based on this model was presented so that agents can increase either utility in negotiations. It was concluded after experimentation that this adaptive negotiation strategy is effective for getting the best negotiation satisfaction degree and both the buyer and seller agents can obtain their maximum utilities with it.

Su *et al*. (2001) described the design and implementation of an internet-based negotiation server for ecommerce and e-business enterprises. The server could be installed by enterprises and it would represent their interests by having their individual negotiation policies and strategies inputted to use to conduct bargaining type negotiations on their behalf. With sellers publishing information about their goods and services online, buyers could identify the seller and if both had installed and registered with a negotiation server, the automated negotiation could begin. This system was found to be a potential alternative to agent-based systems as it makes use of a content specific language, which determines how information about goods and services is published via specifying the data characteristics. This allows buyers to find goods and services from different buyers with whom to begin to negotiate with.

Lau *et al*. (2007) designed and developed a distributed service discovery and an agent based, multi-strategy negotiation system to streamline multi-party ecommerce. Much like the system proposed by Su *et al*. above, it provides a web service to centralise negotiation between buyers and sellers. This system takes it a step further by providing software agents to handle negotiations for both parties. An empirical study was conducted to evaluate such agent's performance. Their performance was compared with that of their human counterparts. The study found that their negotiation agents outperformed their human counterparts in terms of both increased joint payoff and reduced total negotiation time.



Comparative negotiation performance (Agent) vs. (Human)

Negotiation case	(Agent)		(Human)	
	Joint payoff	Time (Rounds)	Joint payoff	Time (Rounds)
1	2.53	15	2.02	22
2	3.06	8	1.98	25
3	3.11	12	1.68	16
4	2.17	14	1.89	33
5	1.98	9	1.33	24

Figure 4 Agent vs Human negotiation system performance for Lau (2007)

Luo *et al*. (2003) argued that the focus of passed research on the topic of automated negotiation has been on the efficiency and effectiveness of the algorithms used to accomplish it (such as above with Lau *et al*.). Lou stated that this is only half the picture. Agents need to be able to represent their owners' preferences to effectively negotiate on their behalf. This requires acquiring knowledge of these preferences. A case study was presented to combat this shortcoming. In it, it was indicated exactly how knowledge for a negotiation algorithm could be acquired. The study identified that user trade-off preferences play a fundamental role in negotiation and found that it is particularly important for negotiation agents to acquire them.

A paper by Yan *et al*. (2007) mirrors Luo *et al*.'s above opinion about the importance of knowledge within automated negotiation. In this paper it is argued that knowledge plays a key role in automated negotiation. Knowledge Beads are suggested to represent knowledge in a way suitable for automated negotiation systems. Knowledge Beads are object-oriented knowledge representation schemes that are encapsulations of definition, behaviour, and data. A knowledge bead can be a composite object or an atomic part object, with each having their own data and methods. Knowledge Beads give a unified approach to representing data throughout the process of the negotiation. It was found that with this unified approach, it allows for stages of the process to be streamlined and data resulted from negotiation to be reused as knowledge in future negotiation.

Badica *et al*. (2006) summarized a state-of-the-art implementation of an automated multi-agent ecommerce price negotiation system that utilized a rule-based approach. They argue that designs for automated negotiation typically follow the same formula of having negotiation protocols ("the rules of the encounter") along with negotiation strategies ("achieve the desired outcome"). In this system, a rule-based approach is used for both. To evaluate the system, an experimental scenario was used which involved multiple buyer agents purchasing items in the style of a standard auction. The results found that rules are a feasible technology for approaching automated negotiation in ecommerce.

Badica *et al*. (2006) presented an update to their initial implementation described above of an agent-based e-commerce system. In this updated system, a Dutch auction rule set was implemented into the negotiations to allow for multiple units of a product to be sold. Within "Dutch" auctions, bidding starts with a starting, extremely high price point. This gets progressively lowered until the item is claimed at the price by a buyer that calls "mine" by pressing a button that stops the clock. This "press and claim" rule set allows multiple units to be sold at once, with the first winner taking their prize and subsequent winners paying less. The researchers claim that combining common traditional auctions and "Dutch" auction rules together could lead to the formation of a set of core (reusable) rules applicable to a wide class of price negotiations.

Yu *et al*. (2008) proposed a multi-agent automated intelligent shopping system to enhance and facilitate transaction and price negotiation within ecommerce. In this distributed system, human buyers and sellers used software agents to undertake the shopping tasks of buying and selling, presenting the results to the respective parties. The system had a multitude of different agents handling various tasks, such as a main agent to handle the creation of buyer/seller agents and various DB agents. The negotiation within the system followed the Dutch auction logic proposed above by Badica *et al.* wherein the price negotiation was continuous until, in this case, the buyer's agent buys the product at the offered price or the seller agent reaches a reservation price.

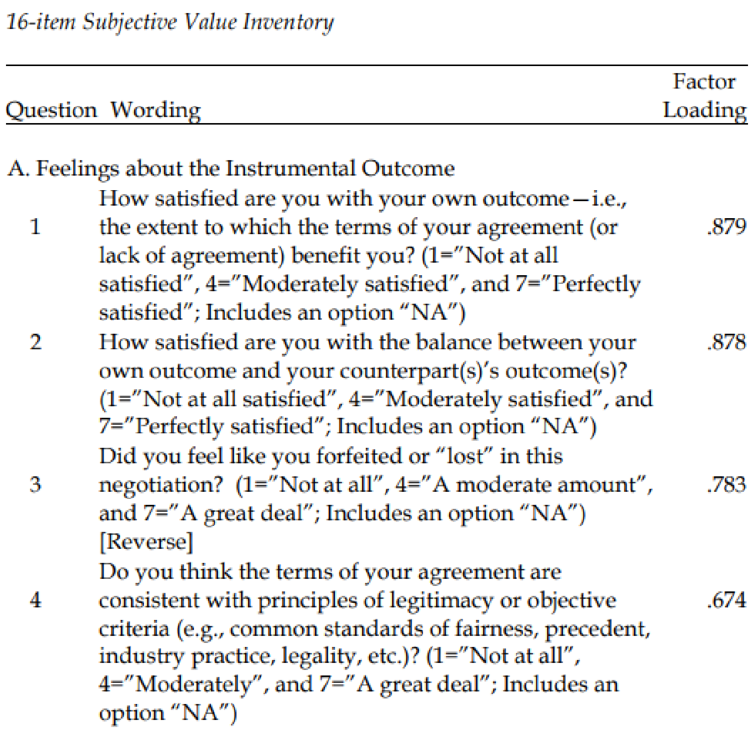
2.6 Satisfaction in Negotiation

In this section, existing research on satisfaction in negotiation will be explored and evaluated to judge the importance of satisfaction with negotiation results.

Najjar *et al.* (2016) proposed a multi agent automated negotiation with an aim of improving user satisfaction within software-as-a-service. The need for improving satisfaction was to combat the findings by Accenture, that stated about 81% of customers switched service providers due to the service not being satisfactory by failing to meet their expectations. The proposed system modelled user expectations and preferences and subjective satisfaction. An autonomous agent represented both end users, the user agent being used to increase the involvement of the user into the decision-making process. The idea being to evaluate whether increased user involvement lead to higher chances of getting a satisfactory service that met their expectations. The proposed approach found that as the integration of user preferences increased, the user's satisfaction also increased.

Schei *et al.* (2005) also mirrored the importance of satisfaction within the realm negotiation when they conducted simulated negotiations with students. They stated that satisfaction is important for a multitude of reasons e.g. in some situations an 'objective evaluation criteria' for negotiation outcomes may not exist and so satisfaction level must suffice. Satisfaction is also important in ensuring willingness to engage in future negotiation with the other parties. The paper stated that previous research in psychology and organizational behaviour showed that single-item measures of satisfaction produce acceptable validity, so to measure outcome satisfaction a post negotiation questionnaire was used, with satisfaction being rated on a 5-point scale. The results of the questionnaire stated that satisfaction was shown to be significantly related to individual outcome and not joint outcome.

Curhan *et al*. (2006) investigated what exactly is valued by people who undertake negotiations. As part of the report a 'subjective value inventory' tool was suggested to systemize and encourage further research on the outcomes of negotiation. Previous examined areas such as trust, satisfaction, and justice were identified as potential variability across negotiations. Satisfaction with a negotiation was identified as a critical element of subjective value. Expectancy and social perception theories were used to argue that satisfaction for negotiators is driven by comparing the expected outcomes of a negotiation to the actual outcomes, a "better than/worse than" heuristic. There are two related values in this framework, satisfaction with the outcome of the negotiation and how much the outcome exceeded expectations. Both these values were used as factors within the SVI tool. A post negotiation questionnaire was used to attain the satisfaction with the outcome value, which was measured via a 7-point scale ranging from “Extremely dissatisfied” to “Extremely satisfied”.



16-item Subjective Value Inventory

Question	Wording	Factor Loading
A. Feelings about the Instrumental Outcome		
1	How satisfied are you with your own outcome—i.e., the extent to which the terms of your agreement (or lack of agreement) benefit you? (1="Not at all satisfied", 4="Moderately satisfied", and 7="Perfectly satisfied"; Includes an option "NA")	.879
2	How satisfied are you with the balance between your own outcome and your counterpart(s)'s outcome(s)? (1="Not at all satisfied", 4="Moderately satisfied", and 7="Perfectly satisfied"; Includes an option "NA")	.878
3	Did you feel like you forfeited or "lost" in this negotiation? (1="Not at all", 4="A moderate amount", and 7="A great deal"; Includes an option "NA") [Reverse]	.783
4	Do you think the terms of your agreement are consistent with principles of legitimacy or objective criteria (e.g., common standards of fairness, precedent, industry practice, legality, etc.)? (1="Not at all", 4="Moderately", and 7="A great deal"; Includes an option "NA")	.674

Figure 5 Part of the post negotiation questionnaire used by Curhan et al.

This same type of post negotiation questionnaire was used in a paper by Yang *et al.* (2009) and once again, echoes the importance of satisfaction. In this paper a model for strategic, automated agent to human negotiation is formalized. The paper integrates insights from psychological and behavioral research with a delayed acceptance and simultaneous offers negotiation strategy to deduce its impact on negotiation outcomes. The paper states that satisfaction is conceivably the most important subjective outcome in negotiation system research, as it reflects the belief that a negotiator has achieved a fair and effective outcome. The satisfaction level of the negotiation outcomes was evaluated using a post negotiation questionnaire. In this, the belief described above is a two-dimensional construct; the negotiators confidence with the solutions settlement efficiency and the perception that the negotiators outcome is fair when compared with his or her opponent.

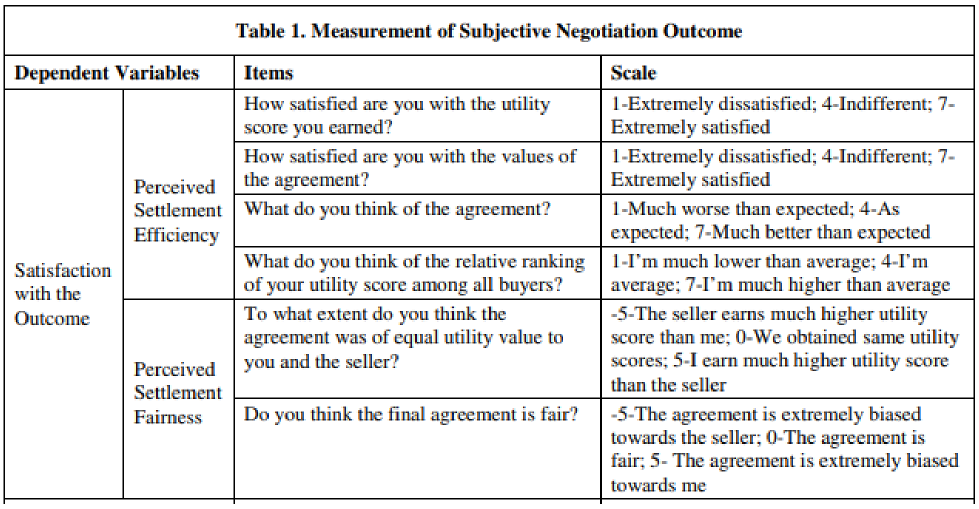


Table 1. Measurement of Subjective Negotiation Outcome

Dependent Variables		Items	Scale
Satisfaction with the Outcome	Perceived Settlement Efficiency	How satisfied are you with the utility score you earned?	1-Extremely dissatisfied; 4-Indifferent; 7-Extremely satisfied
		How satisfied are you with the values of the agreement?	1-Extremely dissatisfied; 4-Indifferent; 7-Extremely satisfied
		What do you think of the agreement?	1-Much worse than expected; 4-As expected; 7-Much better than expected
		What do you think of the relative ranking of your utility score among all buyers?	1-I'm much lower than average; 4-I'm average; 7-I'm much higher than average
	Perceived Settlement Fairness	To what extent do you think the agreement was of equal utility value to you and the seller?	-5-The seller earns much higher utility score than me; 0-We obtained same utility scores; 5-I earn much higher utility score than the seller
		Do you think the final agreement is fair?	-5-The agreement is extremely biased towards the seller; 0-The agreement is fair; 5- The agreement is extremely biased towards me

Figure 6 Part of the post negotiation questionnaire used by Yang et al.

Halpert *et al*. (2010) modelled the negotiation process to find a new way to look at the pre-existing negotiation research. The paper presented a model of the negotiation process and tested it via a series of meta-analysis and follow up path analysis.

The model included seven variables:

- relationship between negotiators
- negotiator goals
- expected cooperation
- cooperation behaviour
- negotiator profit
- satisfaction with the negotiation
- perceptions of the other party

The model had three types of outcomes including profit, negotiator satisfaction, and negotiator perceptions of bargaining opponent. The intercorrelations of the seven variables were used to test the hypothesized model via path analyses. To test satisfaction, profits as well as the behaviors within the negotiation (satisfaction with the negotiation task, one's own performance in the negotiation, and the fairness of the outcome) were considered. The results found that negotiator satisfaction was directly affected by profit, with satisfaction in the experiment being a self-reported measure by the negotiator after the negotiation ceased.

2.7 Conclusions

This chapter began by describing how the combination of the internet and business has created a whole new and rapidly growing form of commerce; eCommerce. With it, came research into the possibility and practicality of supporting negotiations in the context on the internet. Two main approaches were found as ways of enabling online negotiations; automated and non-automated. Non-automated negotiation systems are used to simplify, aid or facilitate the process of negotiation for users. In automated approaches to enabling negotiation, software agents are used in place of and to the benefit of the users within the negotiations. The use of software agents within these systems, regardless of it being automated or not, was found to be of extreme popularity. Automated approaches proved to be the more popular of the two. The inclusion of some form of automation for the seller and buyers using software agents was found to be very common. Within existing automated negotiation systems, it was common to use agents to represent both the buyer and seller. Mutual beneficial outcome was a common talking point and evaluation measure throughout the research, what was uncommon were systems that focused on achieving the best outcome for either the buyer and seller only. There was a clear lack of research into negotiation systems that only represented the sellers. The importance of satisfaction within negotiations was described. The research on satisfaction revealed that negotiation satisfaction relies more on individual outcome (which needs to be a self-reported value) than joint outcome, suggesting once again a need for further research on system that focus on individual outcome and not the joint outcome systems that currently exist. Throughout the research, the idea of using information from one negotiation and applying it in other negotiations, although suggested, was not researched heavily. Research into systems that use data on their customers for negotiation purposes was suggested but once again, not researched heavily.

3 CHAPTER 3. NEGOTIATION SYSTEM DESIGN

3.1 *Introduction*

In this chapter, a new type of negotiation system based on the findings from Chapter 2 will be designed. This automated negotiation system will represent owners of eCommerce stores and using software agents, will negotiate with customers to get the best individual outcome for the seller with each negotiation. The system will use past negotiation and customer data in its strategy to aid in getting the highest sale amount during each negotiation.

First, a software development process will be selected based on researching common methodologies, including iterative models such as waterfall and agile models such as feature driven development. Following this, suitable technologies for the development of the system will be explored, including: Angular, AngularJS, Bootstrap, PHP, NodeJS, Ruby on Rails and MySQL. After which, the architecture of the project can be described by breaking it up into three separate tiers and detailing the technologies which will comprise each tier. Next, the user interface of the system will be prototyped using both paper prototypes and wireframes. At this point, the experiments that will aid in the evaluation of the system from the perspective of both customers and the store owner will be discussed. Finally, the system will be modelled using an ERD (entity relationship diagram), class diagrams and use case diagrams, to provide a foundation for the implementation.

3.2 *Software Development Process*

In this section, the term software development process is defined and the two most common types; iterative and agile, are explained so to make an educated decision on which one to use for the duration of the research project.

3.2.1 Definition

A software development process is the process dividing up the work involved in software development into phases, with the aim of improving the design, development and product/project management. The software process models play a very important role in software development (Kaur, 2013). This process is sometimes known as a software development life cycle. The lifecycle focuses on the product, defining the state through which a product passes from when it starts to be built to when software enters operations and is finally retired (Scacchi, 2002). Numerous approaches to the software development process have been used to structure, describe and prescribe the software development process since the origin of information technology, but these various approaches usually belong to one of two main categories: traditional and agile. Development teams typically pick a software development process from one of the two main categories, though occasionally may combine them.

3.2.2 Traditional Development Processes

Traditional software development processes use pre-organized stages that flow from one to the other in a unidirectional nature. Traditional methodologies are plan driven in which work begins with the elicitation and documentation of a complete set of requirements (Awad, 2005). In the 1960s, "code and fix" was the approach taken by software developers. By the 1970s developers had added more stages to the process, and Winston Royce codified these into a model (that would later become known as the Waterfall Methodology); an approach that Royce warned was "risky" as software testing was only occurring at the later stages of the development process. The Waterfall methodology can sometimes be seen being used interchangeably with "traditional" software development process, due to its popularity within the traditional category. As with other traditional software development processes, waterfall has a set of phases with specific deliverables and documentation. Stage one is to gather the requirements of the system to be developed, the system is then designed, after which the implementation is developed, then tested and finally the system is maintained.

Traditional development processes are best suited for projects with well understood requirements. The requirements of the system need to be given during the very first phase of the project, and new requirements cannot be added beyond this point. Solutions need to be determined early in the process and cannot be changed later down the line. Due to these restrictions, traditional approaches like Waterfall are only suitable when there is no chance of any major changes being made to the system at any point passed the initial stage of the process.

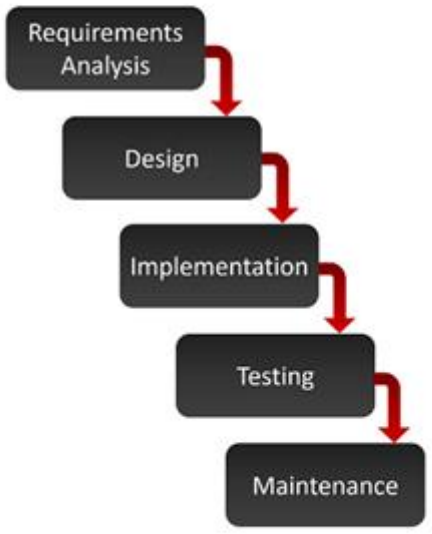


Figure 7 Waterfall software development process

3.2.3 Agile Development Processes

Though waterfall can still be found being used today, developers found it to be frustrating and difficult even back during the 1970s. This resulted in various other software development process methodologies being developed in response to the difficulties faced with traditional approaches like waterfall. These methodologies and practices were based on iterative enhancements, a technique that was introduced in 1975 and later become known as agile methodologies (Awad, 2005).

Agile proposes an iterative, incremental approach to software development. Iterative development breaks the project into iterations of variable length, each producing a complete deliverable and building on the code and documentation produced before it (Cohen, 2003). The use of iterative development is common to all agile methods (Greer 2011). Developers can make changes to the system functionality throughout the development phases of the project. Agile methods aim to answer a need to develop software quickly, in an environment of rapidly changing requirements (Greer, 2011). A dominant idea in agile development is that a team can be more effective in responding to change if it can (Cockburn, 2001).

Advantages of Agile over traditional include:

- Although the software solution must be defined in advance, it can be modified at any stage.
- The solution can be broken down into different modules which can be delivered periodically.
- These modules can be turned into reusable components for future projects.
- Documentation is less of a priority which leads to more development time and less expenditure.

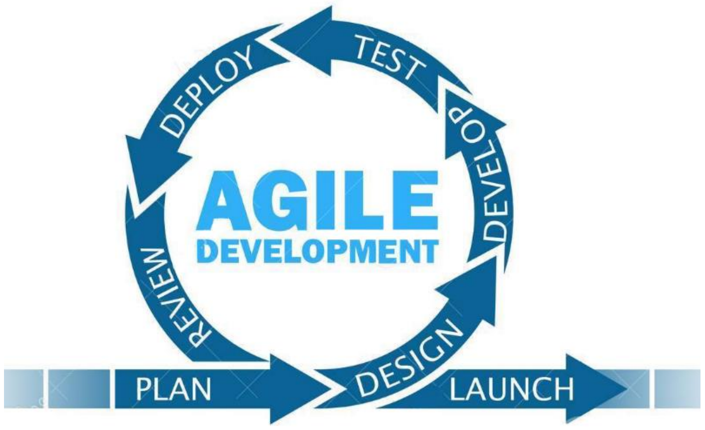


Figure 8 Agile Software Development Process

3.2.4 Selection of Development Process

Deciding whether to utilize a sequential or an iterative development process requires looking at several factors in order to choose the most appropriate approach. An optimal software development process is regarded as being dependent on the situational characteristics of individual software development settings. Such characteristics include the nature of the application(s) under development, team size, requirements volatility and personnel experience (Clarke, 2012).

As the negotiation system passes further through development and more and more aspects of the system are developed, the way negotiation and customer data can be used in tandem will be explored further and new ways to use them may be discovered. The result of which could ultimately lead to more features being developed than originally intended. If a traditional software development process is used and a discovery is made that could have a positive impact on the system, the system will not be able to be modified to utilize what is learned. Methodologies like waterfall do not allow for modifications to the original design and so an agile approach will be better suited for the project. Due to the relatively short development time available for the project, feature driven development seems like the perfect agile approach to take, as the norm for fast-cycle-time projects is a feature-driven iterative process, beginning with features and modelling, followed by design-and-build increments (Benoit, 1999).

Feature driven development, devised by Jeff De Luca in 1997, is an incremental and iterative development process that blends several industry-recognized best practices together from a feature perspective. A feature, as a term, is used for describing a small piece of valuable (sometimes also attractive) capability/functionality (Pang, 2004). Feature driven development consists of 5 activities within a short iteration process. The first two activities being sequential, and the final 3 iterated for each feature.

1. *Develop the overall model* – a walkthrough of the system is undertaken to settle on scope and context and then detailed domain models are created before merging them into one overall model.
2. *Build the feature list* – a list of features is identified by breaking down the system into subject areas. These features should not take more than two weeks to complete or they should be broken down further.
3. *Plan by feature* – a development plan is created, and feature development tasks are assigned to developers.
4. *Design by feature* – each feature has a design package produced which includes sequence diagrams and information that can refine the overall model. Finally, design inspections are held.
5. *Build by feature* – once the design inspection is successful, the code is developed and then unit tested and inspected. This feature then is added to the main build.

3.3 System Technologies

To evaluate the negotiation system from a functionality and suitability standpoint, it needs to be developed and then integrated with an ecommerce site. This website will be developed specifically for the purpose of aiding in the testing and evaluation of the negotiation system. The website needs not have the full functionality that would be traditionally expected to be present within an ecommerce site. The site only needs to allow for users to register and then select a product to negotiate on. Due to the website's simplistic nature and need for bespoke functionality, the site will be developed from scratch. In this section, potential technologies that would be suitable options to develop the front and back ends of both the ecommerce website and the negotiation system will be described. Finally, as in some cases multiple suitable technologies for the project exist, the technology that will be used in the project will be selected so that the final project architecture can be described and explained.

3.3.1 Front End

In this section, suitable technologies for developing the front end will be described.

AngularJS

AngularJS is a front-end web application framework that aims to simplify both development and testing of web applications using a client-side model view controller architecture. AngularJS is not a library rather AngularJS is a JavaScript framework that embraces extending HTML into a more expressive and readable format (Jain, 2015). Mostly maintained by Google, AngularJS aims to address common challenges encountered when developing single paged applications. A Single Page Application (SPA) is composed of individual components that can be replaced or updated independently, without refreshing whole page so that the entire page does not need to be reloaded on each user action (Jadhav, 2015). The purpose behind this is to make the subsequent page loads very fast as compared to traditional Request-Response cycle (Jadhav, 2015).

AngularJS provides many advantages to developers:

- AngularJS only begins evaluating pages at the end of the loading processes so adding bits and pieces of angular on top of existing applications is very easy. Angular is one of the only major front-end frameworks that utilize plain old JavaScript objects (POJOs) for the model layer. This makes it extremely easy to integrate with existing data sources and play with basic data (Jain, 2015).
- AngularJS is a very simplistic framework, it requires nothing more than a basic HTML document to be opened inside a browser to begin using its features. This provides a useful way to quickly create website mock-ups or pieces of functionality. AngularJS is the fastest road for us to implement the simplest website as well as most complicated web applications (Jadhav, 2015).
- Angular has a very interesting and extensible components subsystem, and it is possible to teach a browser how to interpret new HTML tags and attributes (Kozlowski, 2013). Creating these custom attributes and elements allows you to extend the standard HTML vocabulary. With this, you can make custom, reusable components.

AngularJS is a Model View Controller based framework. AngularJS's architecture incorporates the basic principles behind the original MVC software design pattern into how it builds client-side web applications (Jain, 2015).

Model View Controller (MVC) is a software design pattern for developing web applications. It is made up of the following separate parts:

- Model – lowest level, maintains the application data. Responds from requests from view and instructions from controller.
- View – displays the data to the user.
- Controller – the code of the software that controls the model and views interactions between each other. Responds to user input by validating it and then sends instructions to the model

MVC supports 'separation of concerns' as the applications logic is separated from the user interface layer. The controller receives the application requests and interacts with the model to prepare the data that is needed by the view. This data is used by the view to generate the final presentable response.

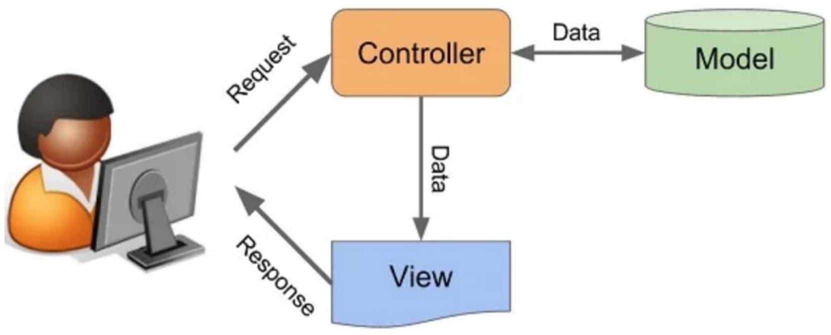


Figure 9 MVC Flow (Schlin, 2019)

The AngularJS architecture is comprised of the following:

1. Root Module
2. Module
3. Config
4. Routes
5. Scope
6. View
7. Controller
8. Factories
9. Directives

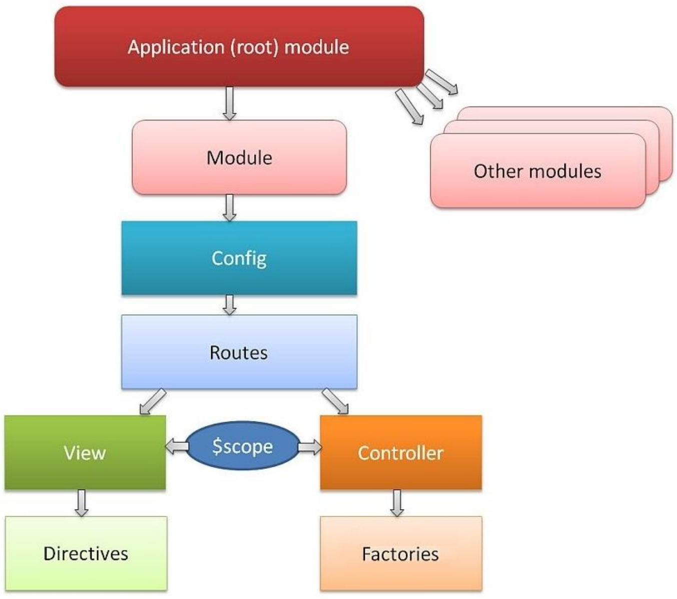


Figure 10 AngularJS Architecture ("AngularJS architecture", 2019)

Module / Root Module	In AngularJS, a module is the way related things can be grouped together. A module, which can be an entire application or a single component of a larger application, is a collection of directives, services and configuration information. Each application must have one module, known as the root module. All others are descendant of this.
Config	Modules, services, providers etc all need to be registered before use within an angular application. This is where config blocks are used.
Routes	Routes allow an AngularJS application to be single pages. Routes are used to move to different pages in the application without the need for reloading.
Controller	AngularJS requires controllers to control the flow of data throughout the application.
View	Where the HTML for the application is contained.
Scope	Scope is an object that binds the view (HTML) and the controller (JavaScript). The scope contains one's available properties and methods for the other.
Factories	A factory is used to return a class or 'service' that can then be injected into and used by modules.
Directives	AngularJS allows developers to extend basic HTML with attributes known as directives. There are built in directives that offer additional functionality to applications. AngularJS also allows for custom directives to be defined.

Table 2 Components of the AngularJS architecture

Angular

Angular (sometimes referred to as Angular 2) is a framework for creating front end web applications. An open source project led by Google, Angular is a ground up rewrite of AngularJS. Although sharing many similarities, major differences do exist. Scope and controllers are not present. It is not an MVC based architecture. Instead, components are used as a primary characteristic of its own differing architecture. Each component has an associated class that handles a specific aspect of the business logic. These components are tied to templates, which are replacement of views in AngularJS. Modules are much more of a core aspect of the framework, having more of the functionality moved to modules than were in AngularJS. No longer is the framework based in JavaScript, Angular uses Microsoft TypeScript language as its native language.

Angular can perhaps be best explained by describing the conglomeration of sub systems and features that make up the framework, these include:

1. Module
2. Template
3. Component
4. Data Binding
5. Metadata
6. Directives
7. Services
8. Dependency Injection

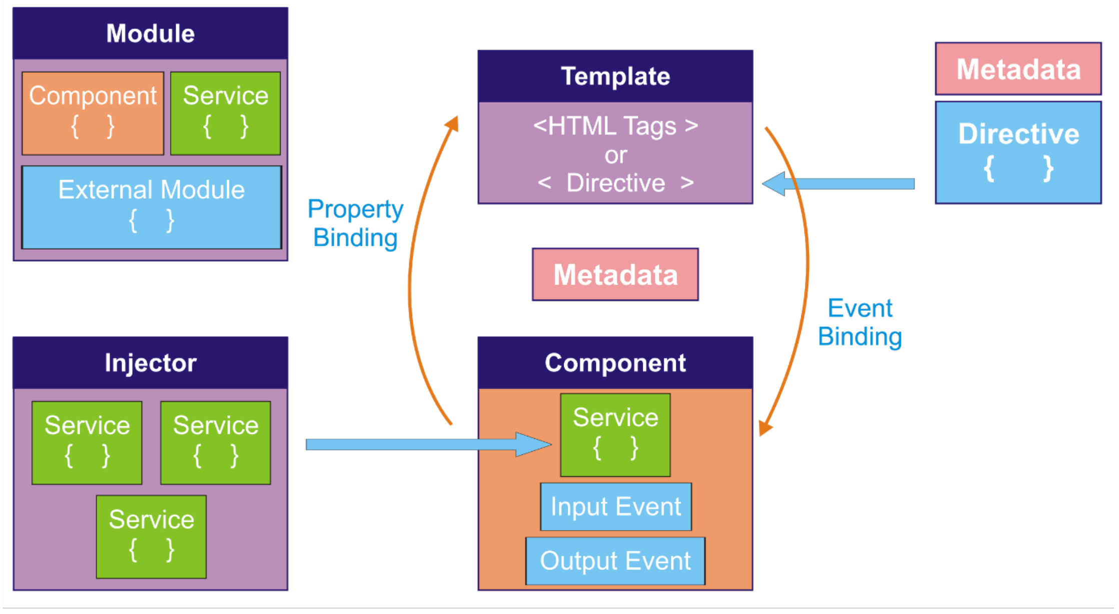


Figure 11 Angular Architecture ("Angular Architecture", 2019)

Module: Angular has its own modularity system to enable the modularity of its applications. This system is called NgModule. Each Angular application has a class with a NgModule decorator to specify that it is the root module (named AppModule). Having these root modules is needed to enable bootstrapping. Bootstrapping is an essential process in Angular, it specifies where the application is loaded. Angular modules help organize the application into connected blocks of functionality and provide compilation context for components.

Template: Templates are a form of HTML that tell Angular how to render components to the screen. Components are each mapped to one template. Regular HTML and templates look a lot like except for a few differences such as directives, data binding, events and component tags.

Component: Components in Angular are the building blocks of the user interface. Components control parts of the screen. Each component is mapped to a template and contain methods, constructors, properties as well as events. Angular applications all have one root component that gets loaded initially and all others are child components of that one root. For example, root component can decide when to load a welcome component. This welcome component can represent a landing page and will have various other child components such as a navbar which will control the login functionality. Angular will create, update and destroy all components as the user moves through the application.

Data Binding: To allow parts of a template to coordinate with the parts of a component, Angular supports various types of data binding.

- Interpolation – used to view a components property on the user interface
- Property Binding – used to update a HTML elements property
- Event Binding – used to generate an event from an element such as when a button is clicked
- Two-Way Data Binding – if a value is updated on screen, two-way data binding ensures it gets updated in the component also and vice versa

Metadata: Metadata is used to ensure Angular knows what type of a class it is dealing with, so to ensure it processes it correctly. Metadata tells Angular how to process its classes by attaching metadata to components and modules, so to tell them apart.

Directive: Templates in are dynamic and so when they are rendered by Angular, the effect they have on the DOM is specified by the instructions given by the directives. There are two types of directives in Angular:

- Structural – change the structure of a DOM template (e.g. ensure the sign in box is only displayed when logged out)
- Attribute – updates an attribute of a specific HTML form for example

Service: Services are pieces of reusable functionality that get shared by components through an Angular application. They can also be used strictly as a data sharing class for components. The most common use for a service is to handle server-side web service calls. Services are asynchronous.

Dependency Injector: When creating a new instance of a class that has dependencies, dependency injection supplies the class with the fully formed dependencies it needs to operate. Most dependencies in Angular are services, so in most cases dependency injection is used to provide components with the services they need.

Bootstrap

Bootstrap (sometimes referred to as 'twitter bootstrap') is a front-end open source framework created to help with the development and design of web applications. Designed at Twitter and released as an open source project on August 19th 2011, Bootstrap become one of the most watched GitHub projects with over 33k watchers (Cochran, 2012). Currently on its 4th iteration, Bootstraps contains an extensive list of features to aid in the development of website front ends, these include but are not limited to:

- Responsive utility classes, allowing content to hide or change size/position dynamically based on the device its being viewed on
- Prestyled components such as buttons, dropdowns, navigation and progress bars
- Templates for rapid development or prototyping

3.3.2 Back End

In this section, suitable technologies for developing the back end will be described.

PHP

PHP (originally standing for Personal Home Page) is a server-side scripting language used for web development. Created in 1994 by Rasmus Lerdof, PHP is most commonly used for producing valid XHTML code and for processing form data submitted by users (Gosselin, 2010). PHP is flexible in that it can be embedded directly into HTML code or used in combination with web frameworks or content management systems on the web. PHP code itself is processed by an Interpreter that is typically a module in the web server. The standard interpreter for PHP is powered by what's known as the Zend Engine. The Zend Engine compiles PHP code into an internal format which it can then execute. The web server can combine the results of the executed and interpreted PHP code, which can then result in the generation of a web page, an image or some other type of data.

The main versions of PHP are:

Version	Release Date	Main Notes
1.0	8th of June 1995	A set of tools to aid in the development of simple yet dynamic web applications through the ability to: • Communicate with databases • Accelerate bug reporting • Extend and embed inside HTML • Handle forms
2.0	1st of November 1997	PHP released as a standalone programming language.
3.0	6th of June 1998	Updated Parser.
4.0	22nd of May 2000	PHP became powered by Zend Engine 1.0
5.0	13th of July 2004	Added greater support for OO programming and added the PDO extension for accessing databases more consistent and efficiently.
6.0	Not Released	Abandoned version.

7.0	3rd December 2015	A more optimized and stabilized version of PHP 5.

Table 3 PHP Major Releases

PHP can be deployed on most web servers and on almost every platform/operating system, all free of charge. This makes PHP a very popular choice for web development, as supported by the fact that it is used by 81.7% of all websites whose server-side programming languages are known (Prokofyeva, 2017).

NodeJS

JavaScript is primarily and typically used for client-side scripting. JS scripts are written and embedded into HTML and then using a web browser, are ran client side by a JS engine. Recently, new technology has started to push the use of JavaScript not only on the client side, but the server side as well. JavaScript has rapidly become one of the most popular programming languages and is now being used in several areas beyond its original domain of client-side scripting (Madsen, 2015).

An example of this is technology is Node.js. Originally written by Ryan Dahl in 2009, Node.js is a JavaScript run-time environment that executes JS code outside of browsers. This enables developers to use JS for server-side scripting, allowing the production of dynamic web pages before the page gets sent to the client's browser. Developers who use Node.js have the advantage of a unified web application, as both the front end and back end can be made with JavaScript instead of having to use separate languages. Node.js popularity surveys performed by official website indicate that the average downloads are over 35,000 since the version 0.10 released in March 2013 (Lei, 2014).

The main features of Node.js include:

- Allows for the creation of web servers.
- Servers are highly scalable without needing threading.
- Inside these servers, "modules" can be created which can handle file IO, networking, cryptography and data streams.

- An event driven architecture which is asynchronous to allow for greater throughput.

Ruby-on-Rails

Ruby-on-Rails is a server-side web application framework. The Rails application-development framework is based on Ruby, an open source, object-oriented scripting language like Perl (Geer, 2006). Released as an open source project in July 2004, it had gained large popularity as by as early as January 2006 the Ruby on Rails development environment had been downloaded almost 230,000 times (Greer, 2006). An MVC based framework, Ruby on Rails provides default structures for such things as databases, web services and web pages. Ruby on Rails encourages the use of standards such as using XML or JSON for transferring data, and HTML, CSS and JavaScript for creating user interfaces. Ruby on Rails emphasizes the use of the software pattern don't repeat yourself (DRY). DRY means writing less code for your application, keeping your code small means faster development and fewer bugs, which makes your code easier to understand, maintain, and enhance. Hibbs (2005).

The main features of Ruby on Rails include:

- MVC based architecture – enabling the separation of data from the presentation.
- Ruby on Rails has a database access library which simplifies the data handling in databases, and maps tables/rows to classes/objects.
- AJAX library functions are provided, and Ruby on Rails code can be used to generate the AJAX code
- Detailed error logs for debugging are provided
- Components can be created to store code to be reused later.

MySQL

Databases can be managed by relational database management systems (RDBMS). An RDBMS supports a database language to create and delete databases and to manage and search data (Williams, 2004). MySQL is an example of one such RDBMS. Development of MySQL began in 1994, with the first version appearing on 23rd May

1995. The name ‘My’ is a combination of the name of Michael Widenius’, the co-founders, daughter. ‘SQL’ is an abbreviation of Structured Query Language. Written in C++/C, MySQL works across many platforms including Linux, Windows, macOS and Solaris to name a few. MySQL is a component in the well-known, open source LAMP (Linux, Apache, MySQL, Perl/PHP/Python) web application software stack. Now owned by the Oracle corporation, the basic version is open source whilst several paid versions are available with additional functionality. MySQL has been used in such popular applications as WordPress and Drupal, whilst also being used in popular websites like YouTube, Google, Facebook and Twitter. MySQL’s architecture is very different from that of other database servers and makes it useful for a wide range of purposes. (Schwartz, 2012).

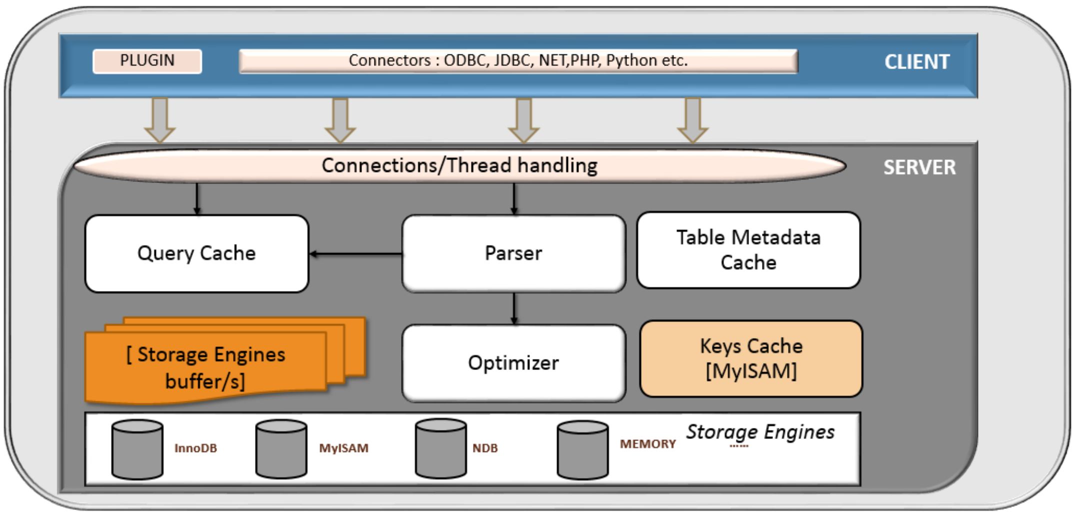


Figure 12 The MySQL Architecture (Lalit, 2019)

1. Client – The utility that connects to the server.
2. Server – The MySQL instance, where the data storage and processing occurs.
3. Connections/Thread Handling – manages the client sessions and connections.
4. Query cache – If a query has been executed, its results are stored in this cache so that if it must be executed again, the results can just be returned in place of executing the query
5. Parser – check the SQL syntax inside the SQL queries
6. Table metadata cache – stores object metadata and information
7. Optimizer – creates execution plans for queries, for added efficiency
8. Storage engine buffer – manages the physical data (file management)
9. Keys cache – stores the table indexes
10. Storage engine – executes SQL statements

3.3.3 Selection

When selecting the technology to use for the project, it seems two options would be most suitable to use in correlation with Bootstrap and MySQL which will be used for certain. Option one is AngularJS with Node.js, and option two is Angular or AngularJS with PHP. AngularJS and Node.js would be a good choice to be used together due to them both being JavaScript based technologies. Using one language across the full software stack would simplify the development process. Despite this, PHP seems like it would be a better choice for the back end over Node.js as PHP integrates nicely with MySQL due to them both being part of the LAMP stack. Angular, being a component-based architecture seems like it would be a better choice over AngularJS. The project involves constructing an eCommerce site and then a negotiation system. Constructing the negotiation system as a component would allow it to be integrated with the Angular website easily.

3.3.4 System Architecture

The negotiation system will be developed with a 3-tier architecture. Using HTTP, JSON messages containing data will be sent from tier 1, the front end, to tier 2, the backend server, and vice versa. Data stored in tier 3, a MySQL database, will be written to and read from by tier 2.

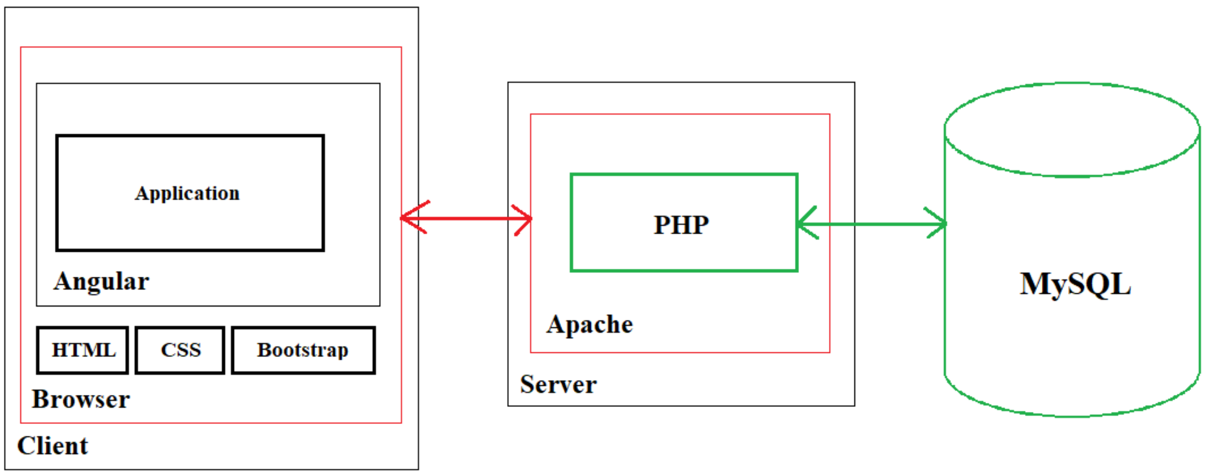


Figure 13 System Architecture

Tier One – The front-end of the system

- Client – The clients in tier one will be the stores customers.
- Browser – The customers will be able to interact with the negotiation system through their internet browser.
- HTML – Will define the structure of the webpage being viewed by the customers.
- CSS – Will help stylise the webpage to the benefit of the user interface.
- Bootstrap – Is a framework that will be used to make the webpage responsive, to allow clients to view the site on various devices.
- Angular – The front-end framework to speed up and enhance the development.
- Application – The webpage will be an angular application, made up of various individual components such as product sections and a navbar.

Tier Two – The back-end of the system

- Server – Will contain the negotiation system and handle all database reading and writing required by the front-end/negotiation system.
- Apache – Will parse the PHP code.
- PHP – The language that the negotiation system will be built with.

Tier Three – The systems database

- MySQL – Relational database management system.

3.4 User Interface Designs

In this section, the user interface for the eCommerce site is designed initially on paper before wireframes are used to create a final design.

3.4.1 Initial Paper Designs

The first versions of the designs for the UI were done quickly on paper so to get an idea of the needed elements and their possible placements.

Welcome Page

The welcome page is the first page the users will see. Below the brand will be two sections, a logging in section on the left, with email, username and password input boxes and a log in button below. On the right, passed some welcome to the site text will be the register section, containing the same input boxes as the log in section and a button below them that allow the users to register with the ecommerce site.

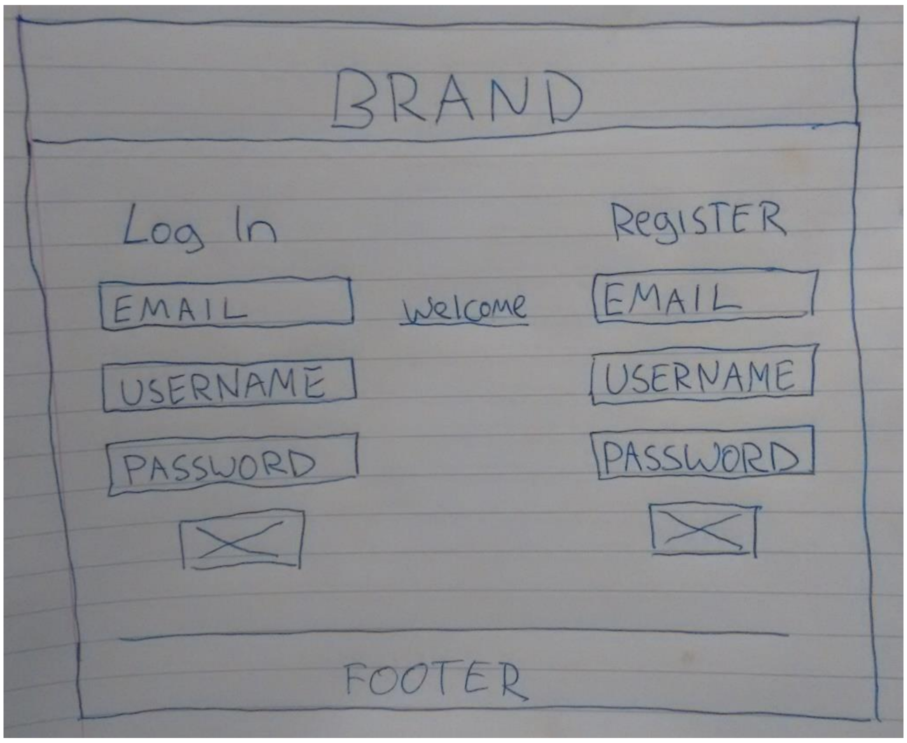


Figure 14 Paper design for welcome page.

Home Page

The home page will be the main page of the ecommerce store. Accessible through log in, users will be able to browse and click negotiate on different products. As this 'store' is needed only to enable testing of the negotiation system, the store will contain the bare minimal needed for this. Only a few different products will be displayed along with their price, image and name. The product type buttons change the category of the products shown on the right of the store. Each product will have a negotiate button (right of price) allowing users to, once clicked, be brought to the negotiation page to use the negotiation system to attempt to agree on a lower price than offered by the store.

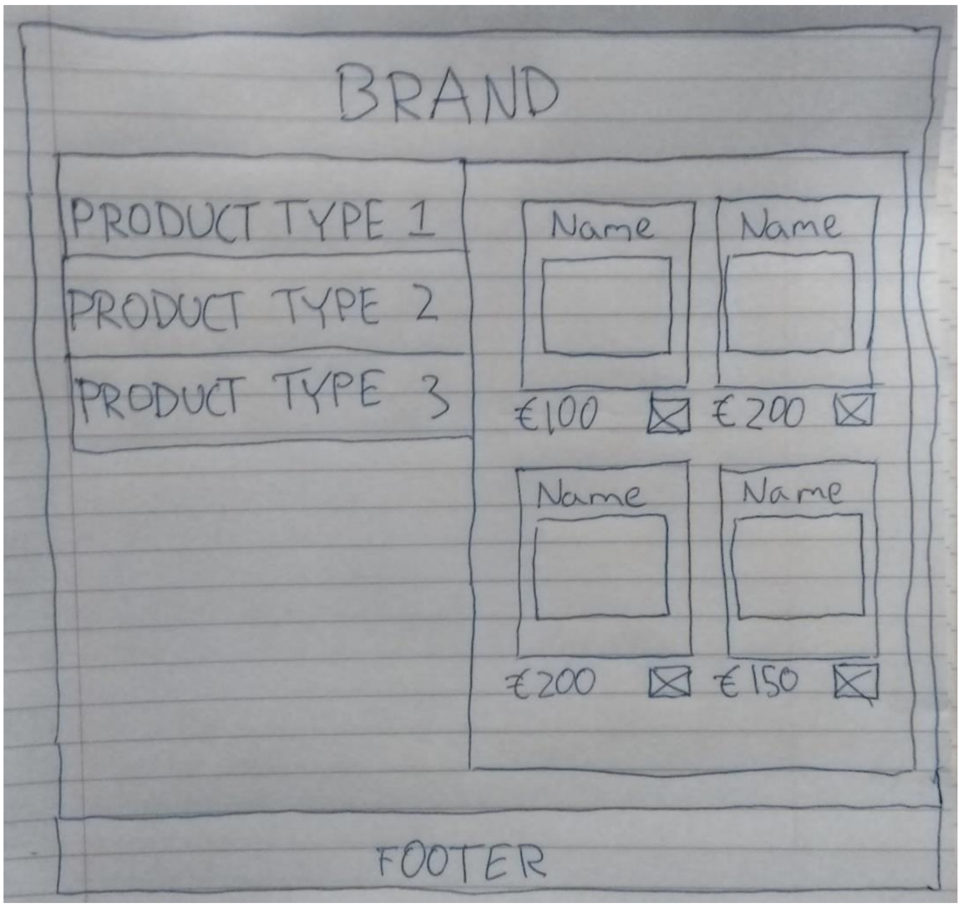


Figure 15 Paper design for home page.

Negotiation Modal

The negotiation modal will pop up when the negotiate button for a product is selected. This modal is where users will interact with the negotiation system. The product the user wishes to negotiate on will be displayed on the right. The current price will be displayed on top, and to its right will be the button to close the negotiation modal. This current price gets updated with each counter offer the system sends to the user, with its final offer causing the 'offer' button to change to an accept button, to accept the deal. Closing the modal ends the negotiation, which also allows the users to reject the final price.

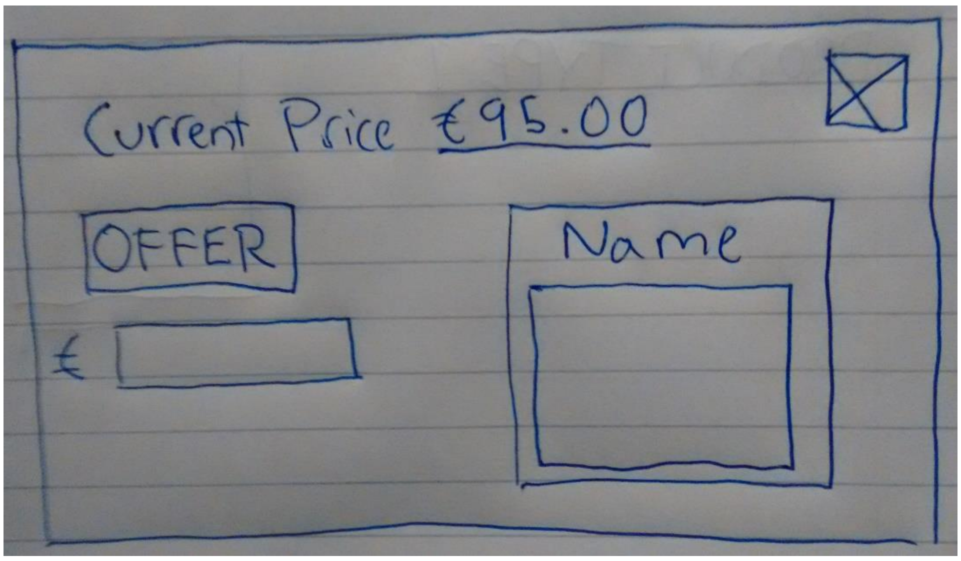


Figure 16 Paper design for negotiation modal.

3.4.2 Refined Wireframe Designs

The following designs are updated and more precise versions of the initial paper-based designs. The application will consist of 3 individual pages. With the use of Angular and its single page application support, the pages will have separate URL's but the transition between them will not require any reloading, the transitions will be seamless. Each page's basic layout will be described using wireframes. The utilization of bootstrap will allow different designs for both desktop and mobile users.

Welcome Page

The updated design for the welcome page will allow users to log into the ecommerce site using the sign in button on the far right of the navigation bar, once their email and password is verified. If the user hasn't created an account previously, they can do so by entering in a unique username along with an email and password. Once the register button is clicked, their account gets created and they are taken to the home page.

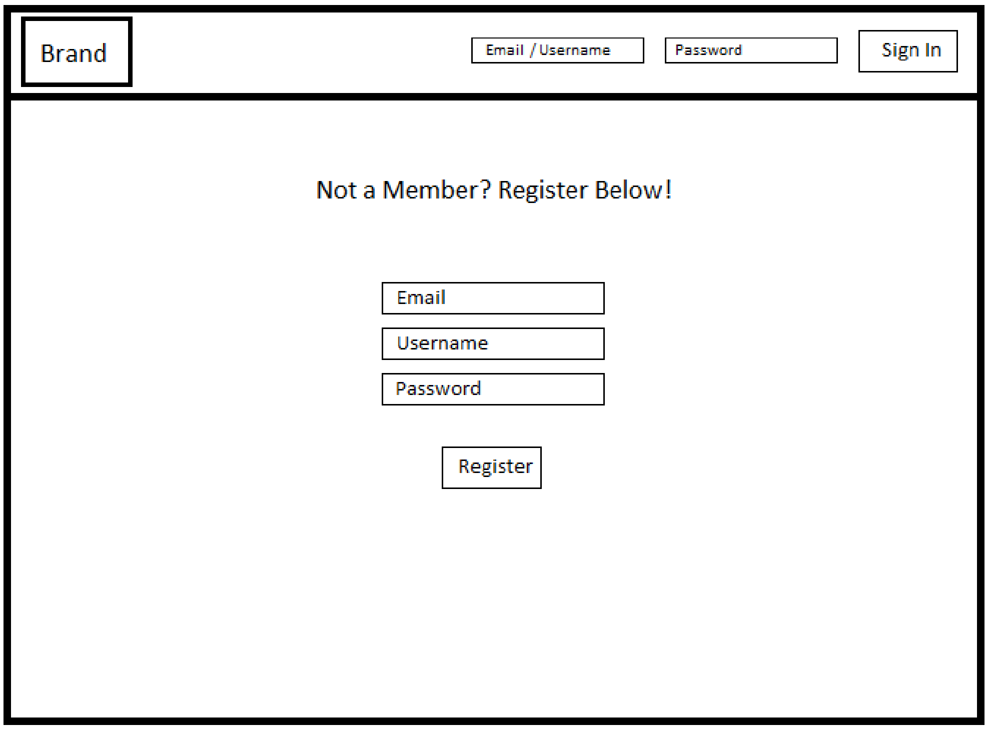


Figure 17 Wireframe for the welcome page

Home Page

The updated version will have no product categories, this will be simplified to just 4 different products in a ‘popular’ section.

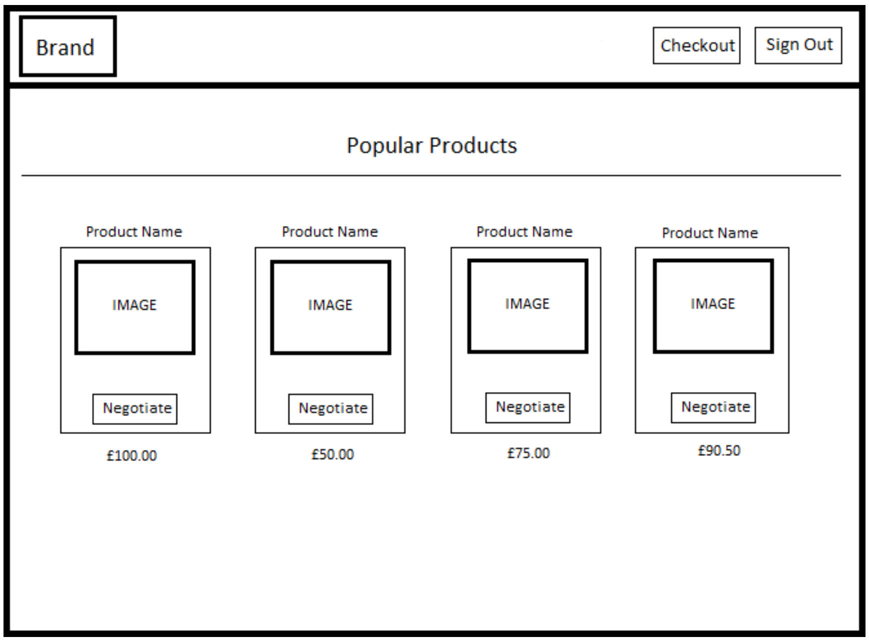


Figure 18 Wireframe for the home page

Negotiation Page

The negotiation page is where users will interact with the negotiation system. The product the customer wishes to negotiate on and its original asking price will be displayed on the left. The current price will be displayed next to the add to cart button, this current price representing what the negotiation system has agreed to let the product go for i.e. its counter offer to the customers current offer for the product. To make an offer, the customer need only adjust the sider to the amount they desire and select the offer button. The negotiation systems counter to the customers offer will be represented by the current price's updated value. If the negotiation system decides its counter Is its final 'take it or leave it' offer, "Current Price" gets replaced with "Final Offer" and the slider/offer button will be disabled.

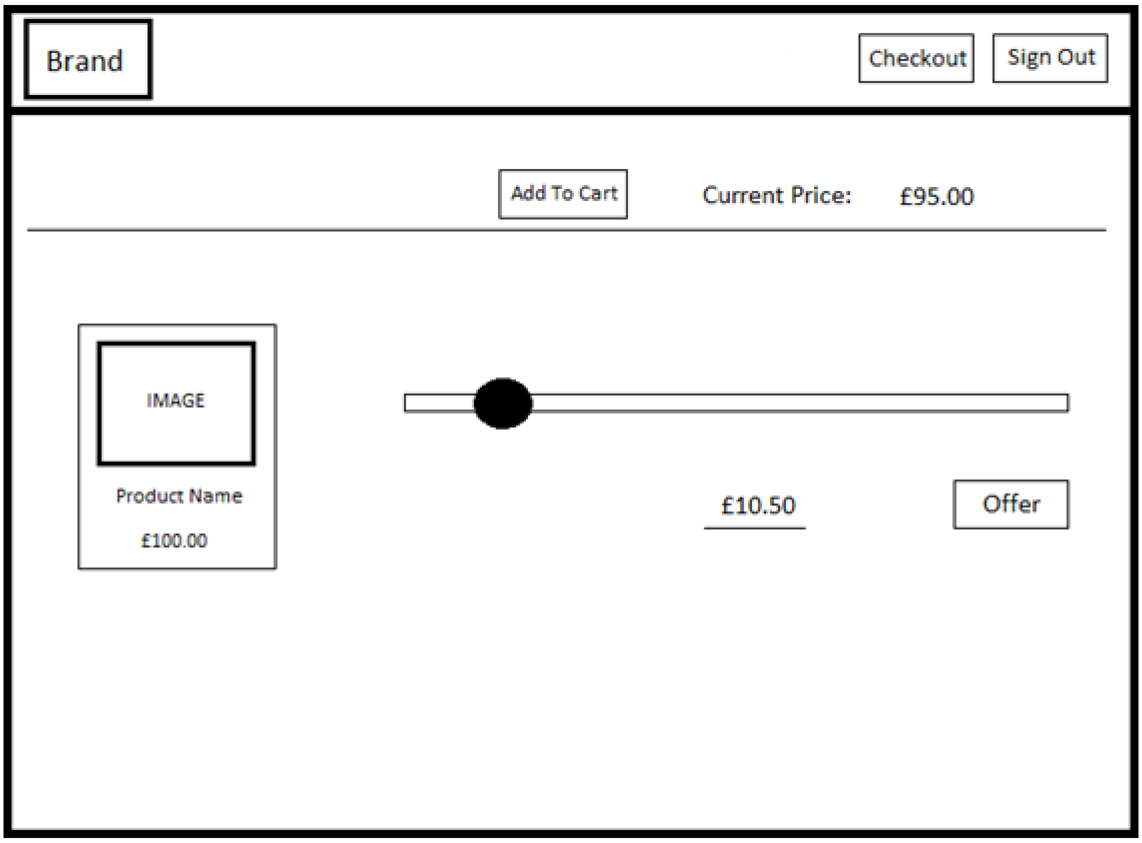


Figure 19 Wireframe for negotiation page

3.5 Experimental Design

The research question pertains to whether the negotiation system can negotiate on a level that is deemed satisfactory to the store owner. Despite this, not only does the store owners satisfaction need to be considered, the customers' does as well. If using the system to negotiate on a price is proved to be a negative experience for customers, then the negotiation system cannot possibly be satisfactory for the store owner. Customers having a negative experience with the system could potentially result in driving away customers. As such, the negotiation system needs to be proved to be an acceptable system to use by customers, to even begin proving it to be deemed a satisfactory system for the store owners. Therefore, two experiments need to take place. The first will experiment will involve the negotiation system itself. The results it can achieve while negotiating with a variety of customers will be evaluated to either support of refute hypothesis 1. The second experiment will involve the customers themselves. The customers experience negotiating with the system will be recorded and evaluated to either support or refute hypothesis 2.

3.5.1 Experiment 1 – Store Owner Satisfaction

The proposed system will exist to aid the store owners, not the customers, in achieving the best sales price possible. With the purpose of the system being to specifically aid store owners, and the fact that negotiation satisfaction was shown to be significantly related to individual outcome and not joint outcome (Schei, 2004), evaluating the system based solely on the results of the negotiations from the perspective of the store owners is acceptable. The final sale price that the negotiation system agrees on with the customer is an acceptable measure of potential satisfaction for store owners as research in the area found that negotiator satisfaction was directly affected by the profit level achieved (Halpert, 2010). Final price / economic gain has been used as an evaluator of negotiator system effectiveness in multiple previous studies such as Ow *et al.* (2014) and Lau *et al.* (2007). The higher the price the negotiation system agrees to sell the products for, the greater the store owners' profit will be. If price is to be used to determine the if the negotiation system is satisfactory for a store owner, then an acceptance zone will have to be used for the price. Although the final price agreed by the negotiator can vary, there will have to be a limit on how low a price the system can offer customers. Negotiation science identifies 'limit' as one of the key standards that drive the negotiator's behaviour (Najjar, 2016). Limit is defined by Pruitt *et al.* (2010) as "a bargainers ultimate position, the level of benefit (or utility) beyond which he or she is unwilling to concede", agreeing on a price below the limit is considered worse than no agreement (Raiffa, 1982). Different store owners may have varied limits. There is no one universal 'limit' that is used by all store owners. As such, the store owners' preference for the limit value needs to be a dynamic within the system.

An experiment will be carried out to evaluate the negotiation systems ability to settle on prices with customers who possess varying negotiating behaviour. If the system is capable of negotiating with customers and selling products to them for prices that are consistently above the limit set by the store owner, then the system can be deemed satisfactory. The negotiation system must be able to obtain satisfactory results even when:

- the number of customers the system must negotiate with is increased
- the limit the negotiation system must stick to is increased
- the number of negotiations the system must engage in with each customer is increased

3.5.2 Experiment 2 – Customer Satisfaction

To evaluate customer satisfaction, an experiment will be held. A group of volunteers will negotiate with the system multiple times on various products. Each volunteer will be interviewed post-experiment in order to gauge their satisfaction with the system. The questions will be based on a combination of the usability heuristics for user interfaces from Nielson (1995) and the post negotiation questionnaires used by Yang *et al.* (2009) and Curhan *et al.* (2006).

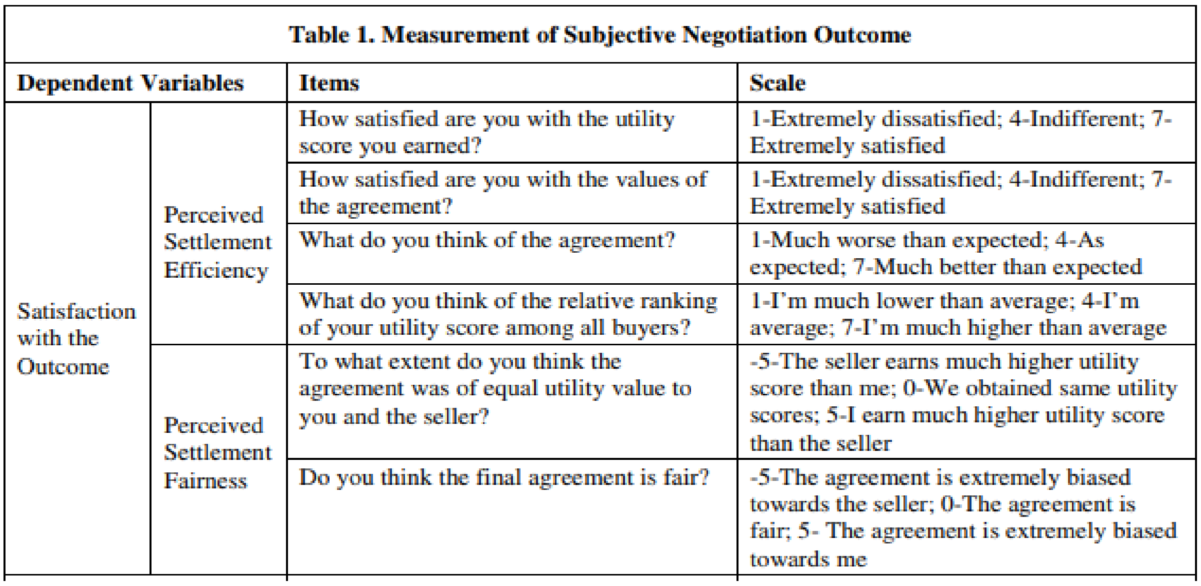
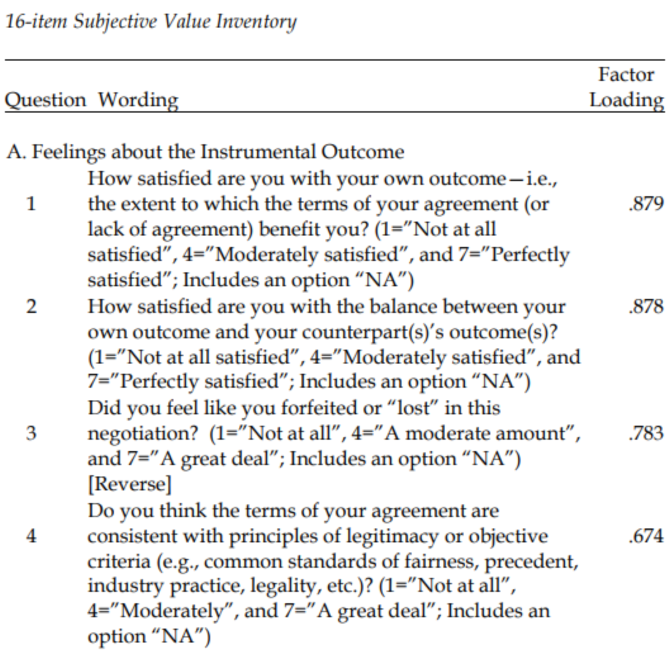


Table 1. Measurement of Subjective Negotiation Outcome

Dependent Variables		Items	Scale
Satisfaction with the Outcome	Perceived Settlement Efficiency	How satisfied are you with the utility score you earned?	1-Extremely dissatisfied; 4-Indifferent; 7-Extremely satisfied
		How satisfied are you with the values of the agreement?	1-Extremely dissatisfied; 4-Indifferent; 7-Extremely satisfied
		What do you think of the agreement?	1-Much worse than expected; 4-As expected; 7-Much better than expected
		What do you think of the relative ranking of your utility score among all buyers?	1-I'm much lower than average; 4-I'm average; 7-I'm much higher than average
	Perceived Settlement Fairness	To what extent do you think the agreement was of equal utility value to you and the seller?	-5-The seller earns much higher utility score than me; 0-We obtained same utility scores; 5-I earn much higher utility score than the seller
		Do you think the final agreement is fair?	-5-The agreement is extremely biased towards the seller; 0-The agreement is fair; 5- The agreement is extremely biased towards me

Table 4 Satisfaction questionnaire used by Yang et al. (2009)



16-item Subjective Value Inventory

	Question Wording	Factor Loading
A. Feelings about the Instrumental Outcome		
1	How satisfied are you with your own outcome—i.e., the extent to which the terms of your agreement (or lack of agreement) benefit you? (1="Not at all satisfied", 4="Moderately satisfied", and 7="Perfectly satisfied"; Includes an option "NA")	.879
2	How satisfied are you with the balance between your own outcome and your counterpart(s)'s outcome(s)? (1="Not at all satisfied", 4="Moderately satisfied", and 7="Perfectly satisfied"; Includes an option "NA")	.878
3	Did you feel like you forfeited or "lost" in this negotiation? (1="Not at all", 4="A moderate amount", and 7="A great deal"; Includes an option "NA") [Reverse]	.783
4	Do you think the terms of your agreement are consistent with principles of legitimacy or objective criteria (e.g., common standards of fairness, precedent, industry practice, legality, etc.)? (1="Not at all", 4="Moderately", and 7="A great deal"; Includes an option "NA")	.674

Figure 20 Satisfaction questionnaire used by Curhan et al. (2006)

3.6 System Design

3.6.1 Use Case Diagrams

When creating use case diagrams for an eCommerce environment, typically a store owner actor would be shown. As the eCommerce site will exist purely to enable the experiments, a store owner actor will not be needed. The products listed on the site do not need to be dynamic, they will be hardcoded. No admin page will exist.

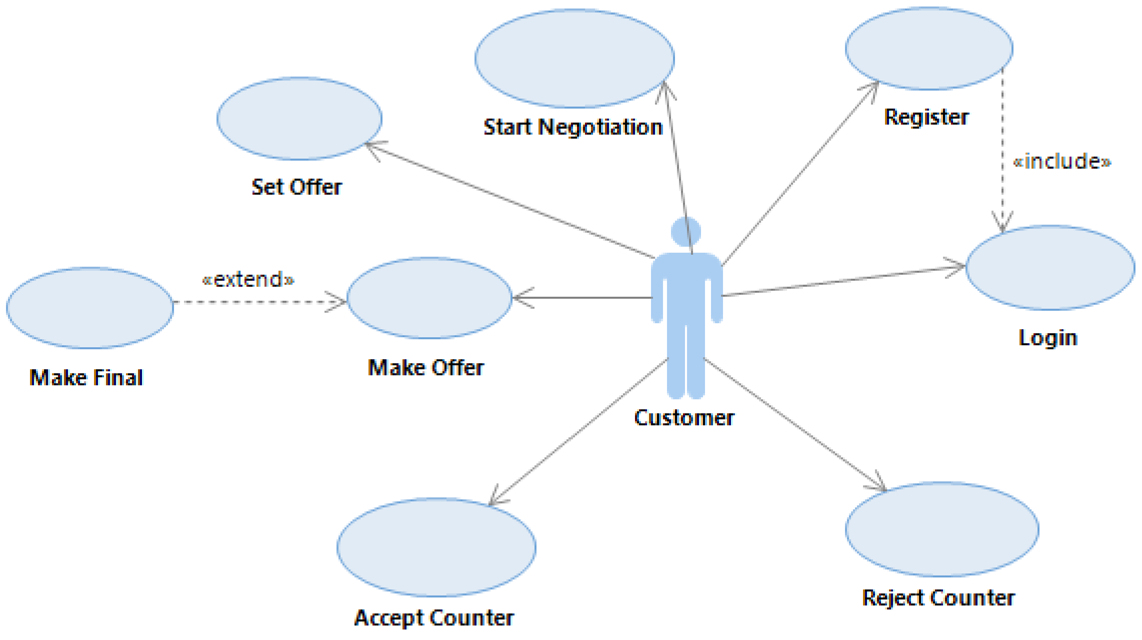


Figure 21 Use Case diagram for customer

When a customer first uses the website, they will be required to register (‘Register’). Once their details have been entered and verified, the log in process is automatically triggered (‘Login’). Any further use of the website only requires logging in (‘Login’). The customer can now hit the negotiation button and start the negotiation (‘Start Negotiation’) for a product. Once begun, the customer can set an offer (‘Set Offer’) to send to the negotiation system (‘Make Offer’). If the offer they wish to send is the max they are willing to pay for a product, the customer can set the offer as their final offer (‘make final’). Once an offer is sent, the negotiation system can either accept it, reject it, or counter it (if it hasn’t been set as the customers final offer). If a counter offer is sent by the negotiation system, the customer can either accept it (‘Accept Counter’), reject it ('Reject Counter') or if the counter offer hasn't been set as the negotiation systems final offer, the customer can set and make another offer.

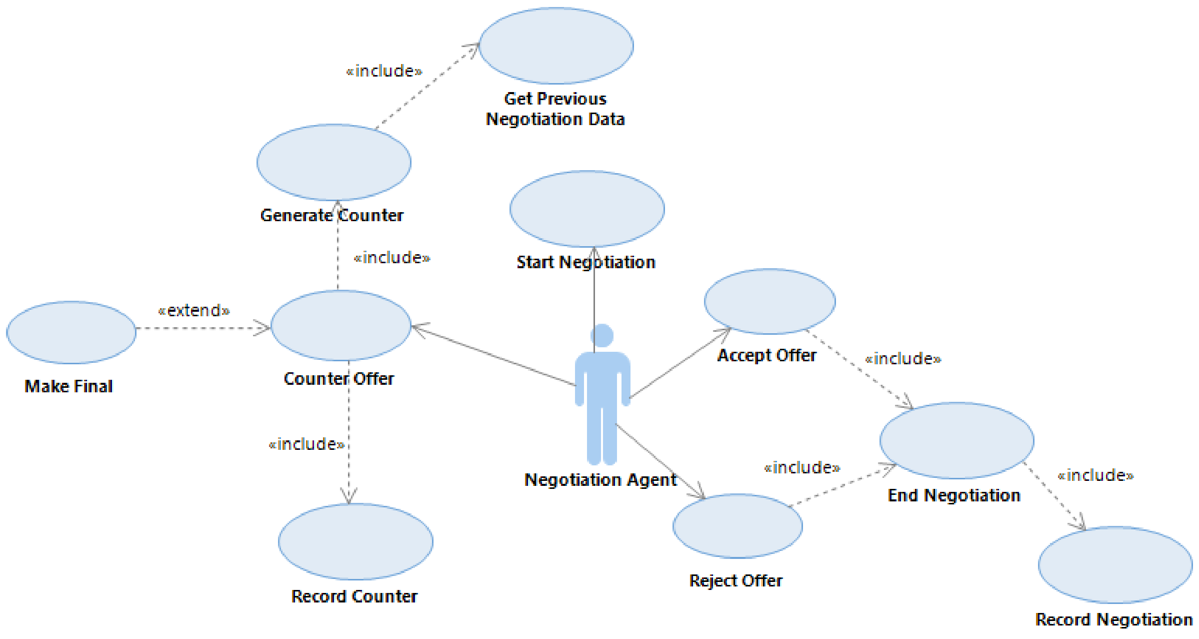


Figure 22 Use Case diagram for Negotiation Agent

Once a customer hits the negotiation button on a product, the negotiation agent starts the necessary preparation ('Start Negotiation') needed to field any and all offers for the product sent by the customer. It will, depending on the offer, either counter it ('Counter Offer'), accept it ('Accept Offer') or reject it ('Reject Offer'). Accepting or rejecting an offer sent by a customer will result in the negotiation agent ending the negotiation ('End Negotiation') and recording the result of it ('Record Negotiation') to use in future negotiations. Countering a received offer involves first generating the counter ('Generate Counter') based on any previous negotiation data on the customer ('Get Previous Negotiation Data'), deciding if the counter is the final counter offer ('Make Final') or not and then recording the counter amount/final status ('Record Counter') before sending the counter offer to the customer ('Counter Offer').

3.6.2 Entity Relationship Diagram

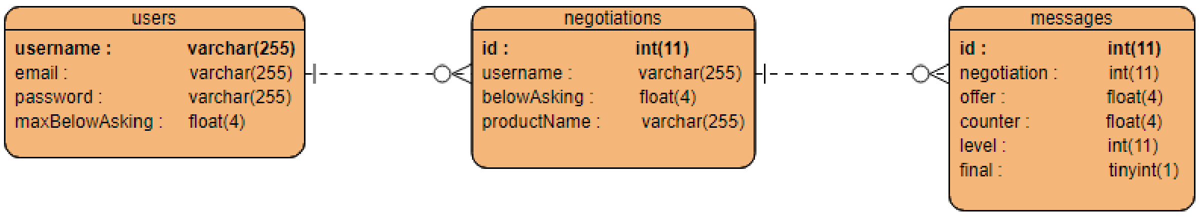


Figure 23 ERD diagram for negotiation system and eCommerce site

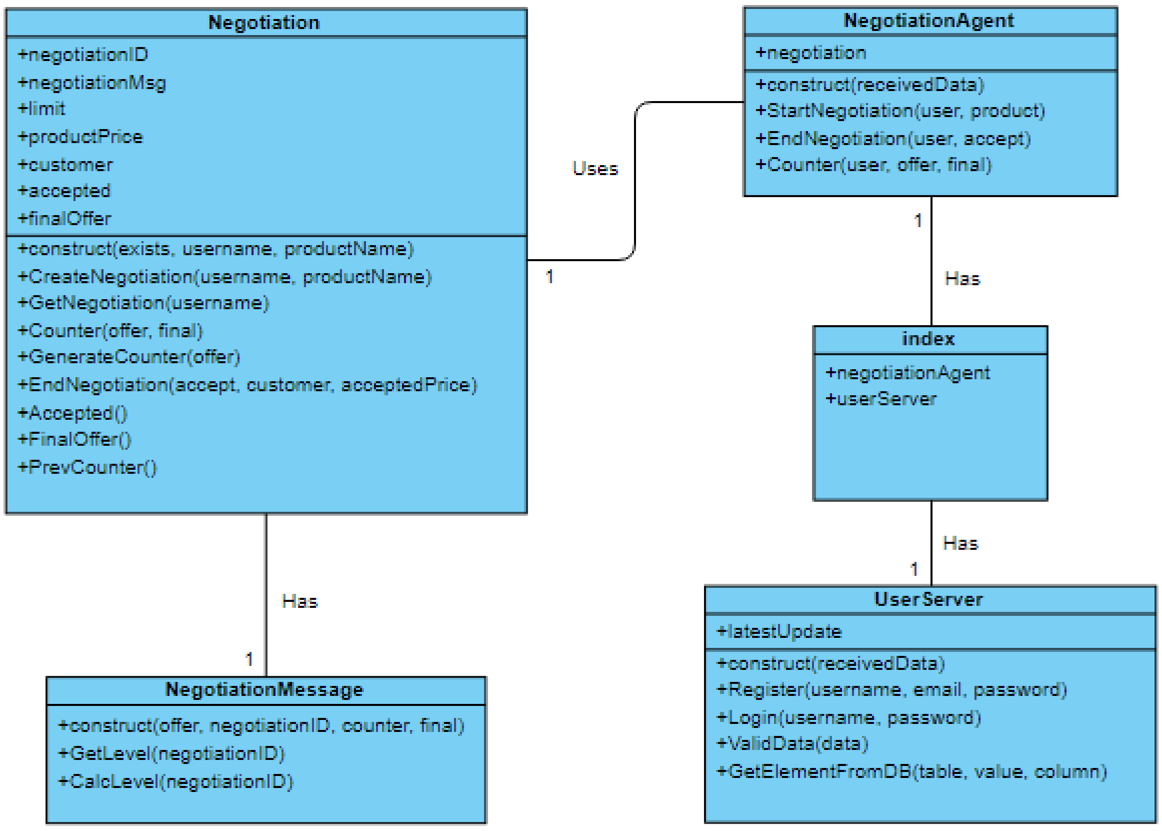
1. Users – stores customer data for registration/logging in and negotiation system decisions
   - username – set during registration, used for logging in and differentiating which negotiation belongs to which customers
   - email – set during registration, used for logging in
   - password – set during registration, used for logging in
   - maxBelowAsking – if a customer sends a final offer, indicating a price is the most they will spend, this price gets converted to a % value below the asking price and gets stored for use in future negotiations (e.g. customer sends 450 as final offer for €1000 product, maxBelowAsking is 45%)
2. Negotiations – stores information for each individual negotiation that occurs
   - id – unique id given to each negotiation that occurs in the system, autoincrements in the database as each subsequent negotiation starts
   - username – the customer who is negotiating with the system, the customer who ‘owns’ this negotiation database object
   - belowAsking – if an agreement is reached on a product between the negotiation system and the customer, the final price agreed by both parties is saved here as a % of the original product price, for use in future negotiations
   - productName – the product the customer and negotiation system are negotiating on
3. Messages – stores each message that makes up each negotiation

- id – unique id given to each negotiation message that occurs in the system, autoincrements in the database as each subsequent message in the negotiation is sent
- negotiation – the id of the negotiation the message belongs to
- offer – the offer sent to the negotiation system by a customer
- counter – the counter the negotiation system has generated in response to the offer
- level – the number of the message in the negotiation (e.g. '2' would represent the second offer/counter message sent between the customer and the negotiation system)
- final – true or false value representing if the message contains a final offer sent by either the negotiation system or the customer

3.6.3 Class Diagrams

The initial class diagram represents a version of the system that can negotiate with customers. This version has a very basic negotiate strategy: just stick to the limit set by the store owner. The final class diagram builds on top of the initial design, allowing the system to negotiate with customers with a more advanced strategy. A negotiation evaluator system will need to be implemented to aid in the evaluation of the negotiation system. This will also be designed with a class diagram.

Initial Design



Negotiation
+negotiationID
+negotiationMsg
+limit
+productPrice
+customer
+accepted
+finalOffer
+construct(exists, username, productName)
+CreateNegotiation(username, productName)
+GetNegotiation(username)
+Counter(offer, final)
+GenerateCounter(offer)
+EndNegotiation(accept, customer, acceptedPrice)
+Accepted()
+FinalOffer()
+PrevCounter()
Uses
1
NegotiationAgent
+negotiation
+construct(receivedData)
+StartNegotiation(user, product)
+EndNegotiation(user, accept)
+Counter(user, offer, final)
1
Has
index
+negotiationAgent
+userServer
Has
1
UserServer
+latestUpdate
+construct(receivedData)
+Register(username, email, password)
+Login(username, password)
+ValidData(data)
+GetElementFromDB(table, value, column)
Has
1
NegotiationMessage
+construct(offer, negotiationID, counter, final)
+GetLevel(negotiationID)
+CalcLevel(negotiationID)

Figure 24 Initial class diagram for negotiation system

Index: The index class will be the entry point for the system. This class will receive JSON messages from the front end and forwards the data to the appropriate class. It contains instances of two classes; UserServer and NegotiationAgent. Index will forward the received messages data to the class it is meant for, e.g. a request to log in and the associated credential data will be sent to the UserServer and will trigger the Login class.

UserServer: This class will handle all registration and log in logic. It will validate all the user credential data it receives and maintain an update variable to contain information such as an incorrect password being used or registration being successful. This variable will be returned to the front end by the index class.

NegotiationAgent: This class represents the negotiation system agent. This agent receives 3 types of negotiation message from the index class. One that indicates a customer wants to negotiate, which triggers the agent to start the negotiation. One that indicates a customer has ended the negotiation, which triggers the agent to do the same. Finally, a message containing an offer on a product from a customer will prompt the agent to counter it. With each message that is received, a negotiation object gets instantiated so to contain the negotiations current state, as each message will affect a negotiation, having an object to represent the negotiation in its current state is needed.

Negotiation: This class represents a negotiation, with each negotiation having a unique negotiationID. As PHP has no state, the negotiation class will get reinitialised after each negotiation message is received by the negotiation agent. This negotiation class will represent a single negotiation in its current state, before the message updates the negotiation (ends it, adds an offer/counter). If the negotiation is just beginning, the class will create a negotiation in the database. When updates are being made to a negotiation, the negotiation class gets the current state before updating. The act of updating creates a negotiation message instance, so to ensure whatever update is represented accordingly in the database. The negotiation class also stores the limit variable, which determines how low the system can let its products go for.

NegotiationMessage: Whatever changes the negotiation class makes to a negotiation, based on the negotiation message sent to Negotiation Agent, the negotiation message class stores this message in the database. For example, a message is sent to the Agent that contains an offer of 100. The agent generates a counter of 90. This offer and counter are part of a single negotiation message. The negotiation message class takes this information and stores it in the database.

Final Design

The final class diagram has updated the initial design in the following ways:

- Added a PastNegotiations class.
- Removed PrevCounter operation from Negotiation and gave it to PastNegotiations.
- Gave Negotiation 2 new operations (SetFinalPrice, RecordMax)

- Gave Negotiation 3 new attributes (max, goal, increase)

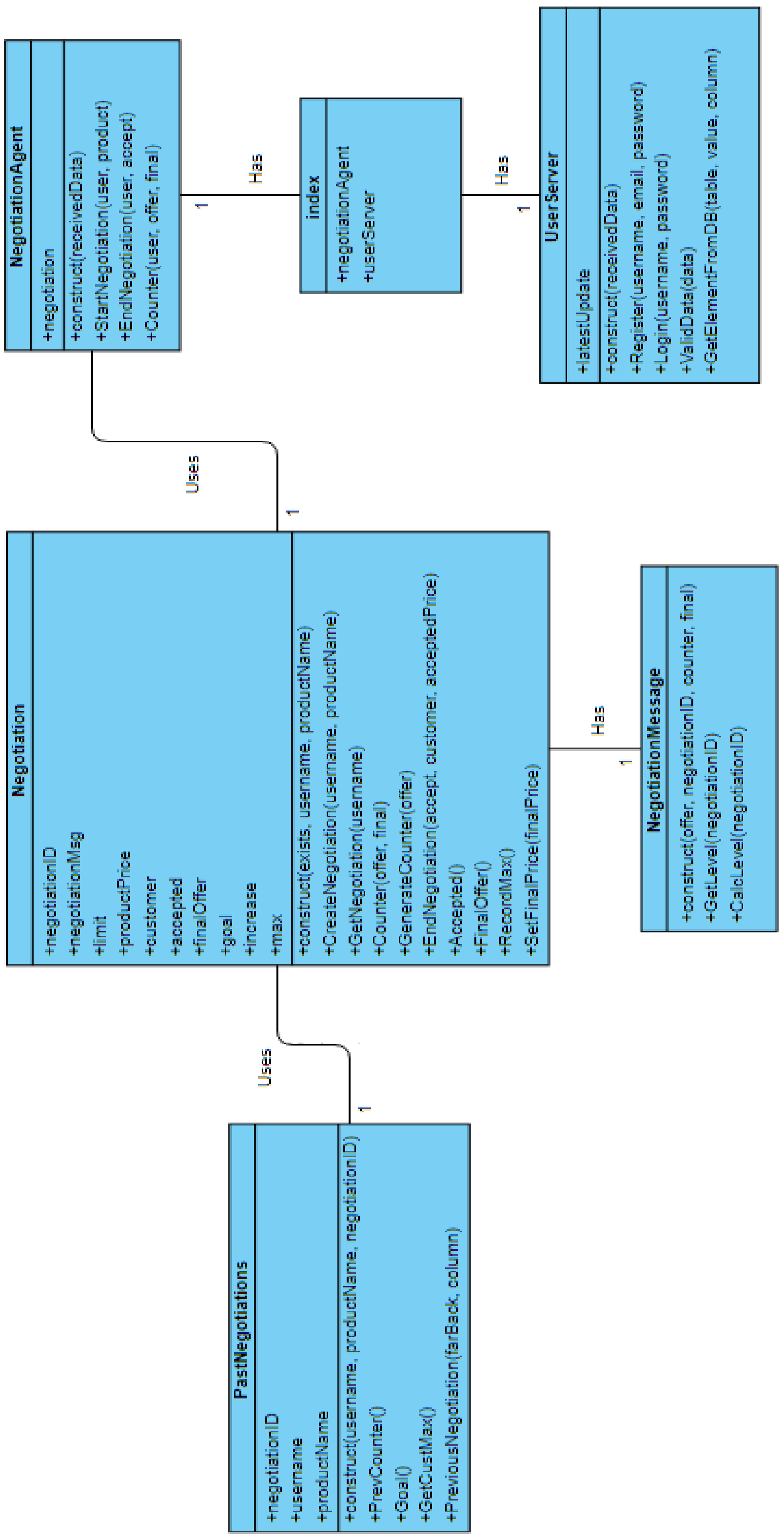

NegotiationAgent
+negotiation
+construct(receivedData)
+StartNegotiation(user, product)
+EndNegotiation(user, accept)
+Counter(user, offer, final)
Has
1
index
+negotiationAgent
+userServer
Has
1
UserServer
+latestUpdate
+construct(receivedData)
+Register(username, email, password)
+Login(username, password)
+ValidData(data)
+GetElementFromDB(table, value, column)
Uses
1
Negotiation
+negotiationID
+negotiationMsg
+limit
+productPrice
+customer
+accepted
+finalOffer
+goal
+increase
+max
+construct(exists, username, productName)
+CreateNegotiation(username, productName)
+GetNegotiation(username)
+Counter(offer, final)
+GenerateCounter(offer)
+EndNegotiation(accept, customer, acceptedPrice)
+Accepted()
+FinalOffer()
+RecordMax()
+SetFinalPrice(finalPrice)
Has
1
NegotiationMessage
+construct(offer, negotiationID, counter, final)
+GetLevel(negotiationID)
+CalcLevel(negotiationID)
Uses
1
PastNegotiations
+negotiationID
+username
+productName
+construct(username, productName, negotiationID)
+PrevCounter()
+Goal()
+GetCustMax()
+PreviousNegotiation(farBack, column)

Figure 25 Final class diagram for negotiation system

The final design of the negotiation class allows it to use the results of previous negotiations to aid future negotiations. Using the PastNegotiations class, Negotiation can get access to the previous settled amount with a customer so that it can add an increase amount on to it to form a goal for the current negotiation. Using PastNegotiations to get the customers max can allow Negotiation to get a max value, this value is the max a customer is willing to pay during a negotiation. If this value is set, then the Negotiation can use it to always get the most it can out of a customer during a negotiation.

Negotiation Evaluator Design

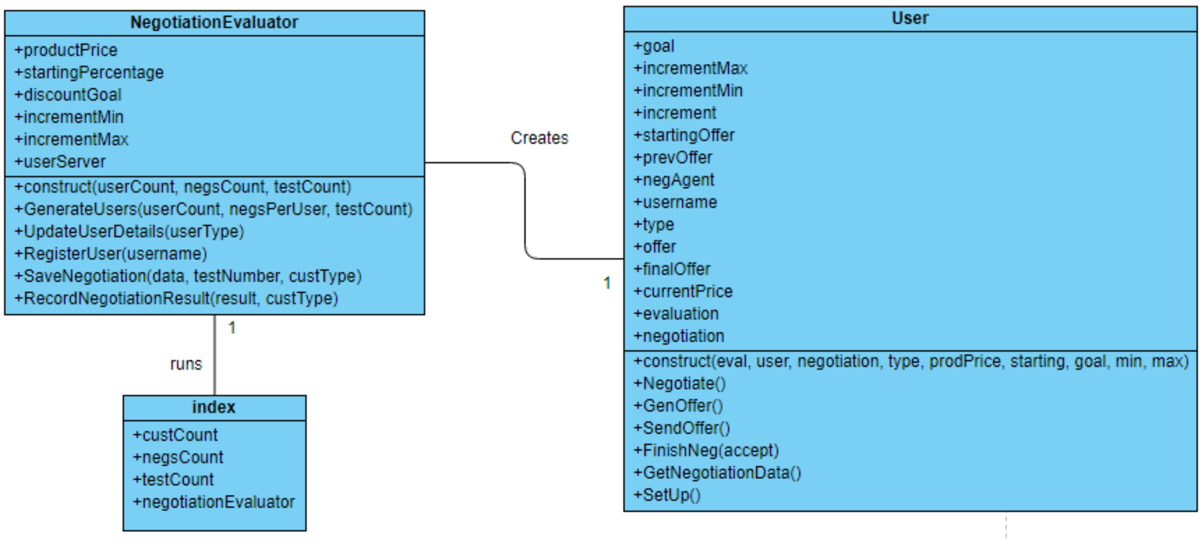


Figure 26 Class diagram for negotiation evaluator

Index: The index class is the entry point for the evaluation system. It receives data from a front end ‘evaluation’ page that contains the number of customers (custCount) the negotiation system will have to negotiate with, the number of negotiations the system will have to engage in (negsCount) with each customer and the amount of times the test should be ran (testCount). An instance of the NegotiationEvaluator class is created and these 3 variables are passed to it.

NegotiationEvaluator: The NegotiationEvaluator classes responsibility is to create the requested number of users for each of the customer types and have them each negotiate with the system the specified number of times. The system will run the evaluation the amount of times specified by the testCount variable. The customer types, of which there will be 16, will each have different negotiating behaviour. The NegotiationEvaluator has one instance of a UserServer (detailed in **3.3.6.1**) to register the users. The class has one instance of the User class that gets reinitialised for every new customer, with the customers unique negotiation behaviour specified by the UpdateUserDetails method. The user class instantiates the NegotiationAgent (detailed in **3.3.6.1**) to use it to negotiate. After every completed negotiation, its results are saved to a text file.

3.7 Conclusions

In this chapter, the proposed automated negotiation system for store owners was designed. This system will represent the store owners using software agents and will negotiate on their behalf with their customer with the goal of achieving the best individual outcome for the seller with each negotiation. The system will use past negotiation and customer data in its strategy to aid in getting the highest sale amount during each negotiation. First, feature driven development was chosen as the software development process for the project. Using a software development process like FDD allows for each new feature to be tested after implementation, rather than at the end of development when the while system is complete. This may potentially lead to some interesting findings as the negotiation system is brought from using a simplistic strategy to making use of the past negotiation and customer data. Angular and PHP were chosen as the main development technologies due to Angulars component-based architecture and PHPs support for MySQL through PDOs. How they fit together was described in a project architecture diagram so to get a better understanding how they will interact with each other. The projects user interface was prototyped using wireframes and the negotiation system itself was modelled using a mixture of use case, ERDs and class diagrams so to provide a foundation from which the system could be implemented in chapter 4. Finally, the experiments that will be used to evaluate the proposed system were described. Two experiments, one to gauge the customers satisfaction with the system and another to gauge the store owner’s satisfaction with the system, will be used to gather results for evaluation.

4 CHAPTER 4. SYSTEM IMPLEMENTATION AND EXPERIMENTS

4.1 Introduction

In this chapter, the front-end (the ecommerce store and negotiation system UI) and the back end (the negotiation system itself) will be implemented based on the designs shown in Chapter 3. The implementation must exist in the context of an eCommerce site and use a software agent to automate the negotiation on the prices with the customers. The software agent will represent the store owner and use a combination of customer and past negotiation data to in its negotiations so to achieve prices that are to the satisfaction of the store owner. Following this implementation, the development of a Negotiation Evaluator system to aid in the evaluation of the negotiation system will be described. This evaluator will be used during the main experiment to generate large datasets of negotiations from which to evaluate the system. The evaluator will also be used through the implementation of the negotiation system to test each new feature to ensure it is working correctly. The results these tests generate will also be used for evaluation purposes further down the line. How each of the two experiments were implemented will also be detailed. Using the two separate experiments allowed for both quantitative and qualitative analysis. Finally, the development process will be reflected on. Any problems encountered during the development will be discussed and the technology used will be judged on its effectiveness during implementation.

4.2 Front-End Implementation

The front-end implementation involves creating the welcome, home and negotiation pages described in the previous chapter. The front-ends messaging capabilities must also be implemented to allow data to be sent to and received from the back-end.

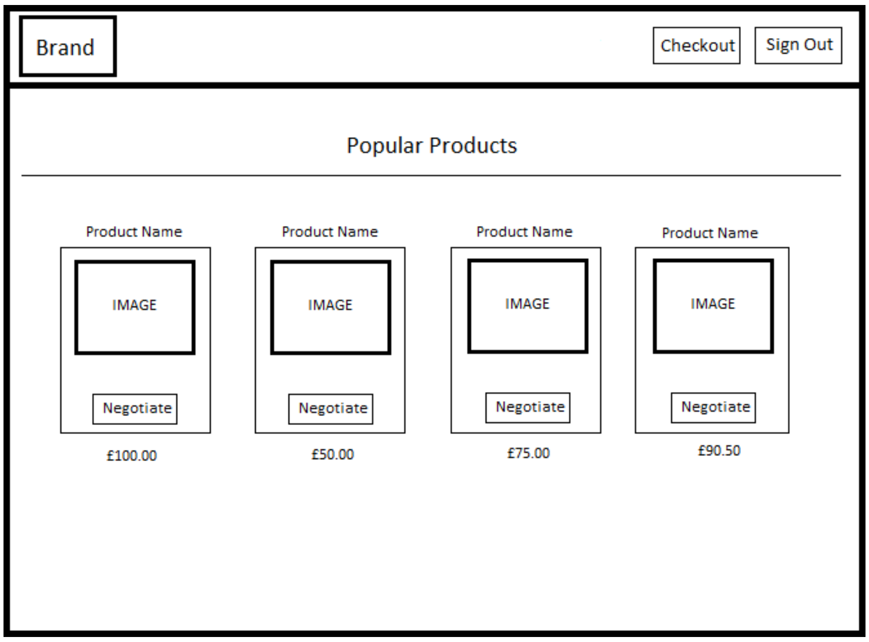


Figure 27 Wireframe for the home page

4.2.1 Front-End Project Structure

To develop the front-end, the Visual Code integrated development environment was used. Angular has a tool called the 'CLI' (command line interface) which allows developers to create new Angular Applications, add services/components/routes and run the application all through simple commands. Using the command CLI command 'ng new ecommerce', a new Angular application called 'ecommerce' was created. This was then opened and inside Visual Code. The entirety of the front-end is an Angular application, with most of the front-end system being developed as angular components. Adding components and services to the project was easy due to Visual Code having plugins specifically for Angular.

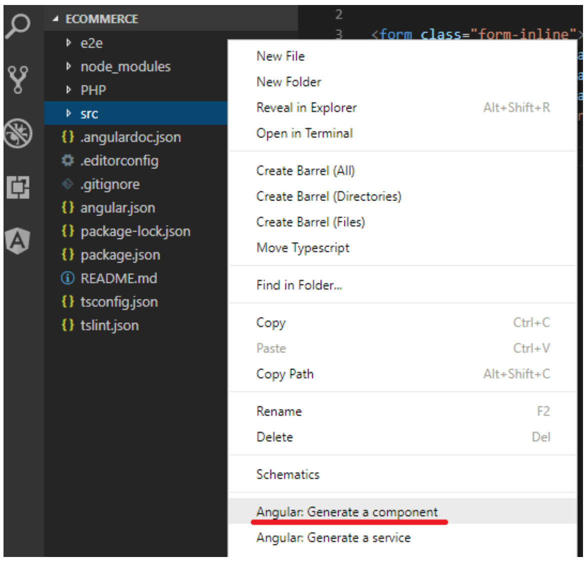
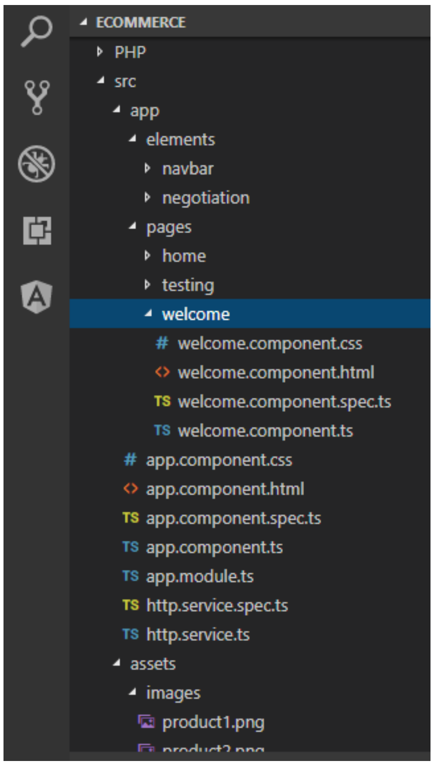


Figure 28 Right clicking on 'src' to add a component using Visual Code Angular plugin

After adding five components and one service, the project structure for the front-end was organised as follows: one Angular app ('app') with two sub sections, one for pages and one for page elements. 'Pages' contains a home and welcome folder, each containing their respective component. 'Elements' contains a navbar and negotiation folder, both of which are also components. Each Angular component has one CSS, HTML, typescript and testing file, all autogenerated on creation and stored in their respective folder. One service was created, named http, to allow communication with the back-end negotiation system. A testing folder and corresponding component were also created (See *Section 4.4* for details). The product images for the ecommerce site had to be stored in the assets folder, as this is where all assets must be stored for an Angular application.

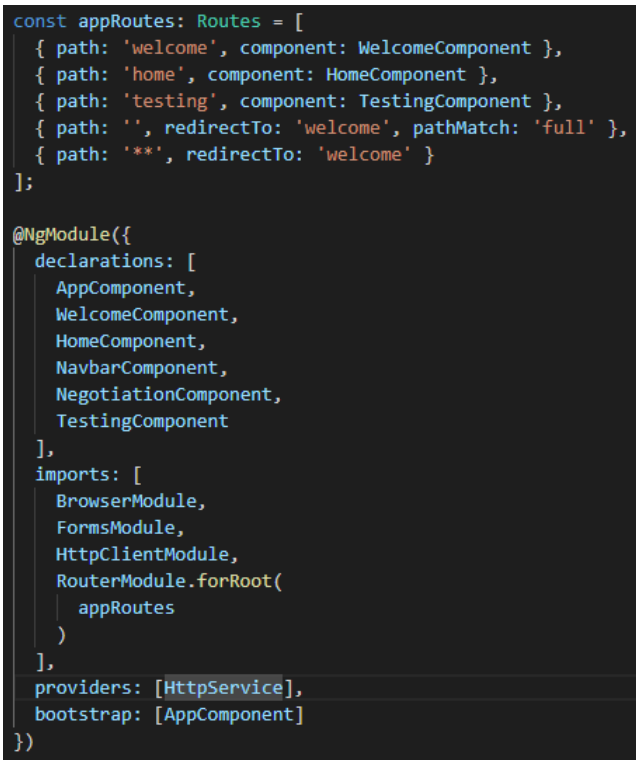


```
ECOMMERCE
  PHP
  src
    app
      elements
        navbar
        negotiation
      pages
        home
        testing
        welcome
          welcome.component.css
          welcome.component.html
          welcome.component.spec.ts
          welcome.component.ts
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
      http.service.spec.ts
      http.service.ts
    assets
      images
        product1.png
```

Figure 29 Angular Ecommerce store project structure

Every Angular component must be declared in the app module class. Here, components are imported, declared, and added to the project's routes array. In routes, each component is given a path (a name which when added to the URL, loads the content of the component without reloading). The base path is set to the welcome component, as this is the first page customers should see on the store. The HTTP service was added to the providers, so it could be used throughout the project.

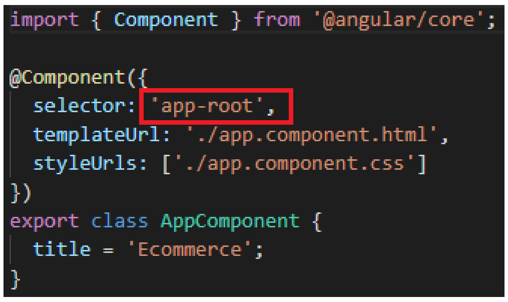
4.2.2 Angular Specific Classes



```
const appRoutes: Routes = [
  { path: 'welcome', component: WelcomeComponent },
  { path: 'home', component: HomeComponent },
  { path: 'testing', component: TestingComponent },
  { path: '', redirectTo: 'welcome', pathMatch: 'full' },
  { path: '**', redirectTo: 'welcome' }
];

@NgModule({
  declarations: [
    AppComponent,
    WelcomeComponent,
    HomeComponent,
    NavbarComponent,
    NegotiationComponent,
    TestingComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    RouterModule.forRoot(
      appRoutes
    )
  ],
  providers: [HttpService],
  bootstrap: [AppComponent]
})
```

Figure 30 app module class

Each Angular application has an app-root. An app-root is the entry point for the application, this is the first component loaded up when the site is entered. This 'app-root' is auto generated when the project is created, it gets, like all other components, a CSS, HTML, and typescript file. In the app.component.ts typescript file, you can see where it is declared as the app-root.



```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Ecommerce';
}
```

Figure 31 app.component class

The HTML for the app root is empty apart from one line: '<router-outlet></router-outlet>'. The router outlet tag is used in Angular to control the flow between the components. The router-outlet displays whatever component the URL path is tied to.

For example, in the routes array in the app module class, the welcome component is tied to the URL '/welcome'. As the default URL in routes is specified to go to the 'welcome' path, the component associated with that path in the routes array, in this case the welcome component, gets displayed.

4.2.3 Welcome Page

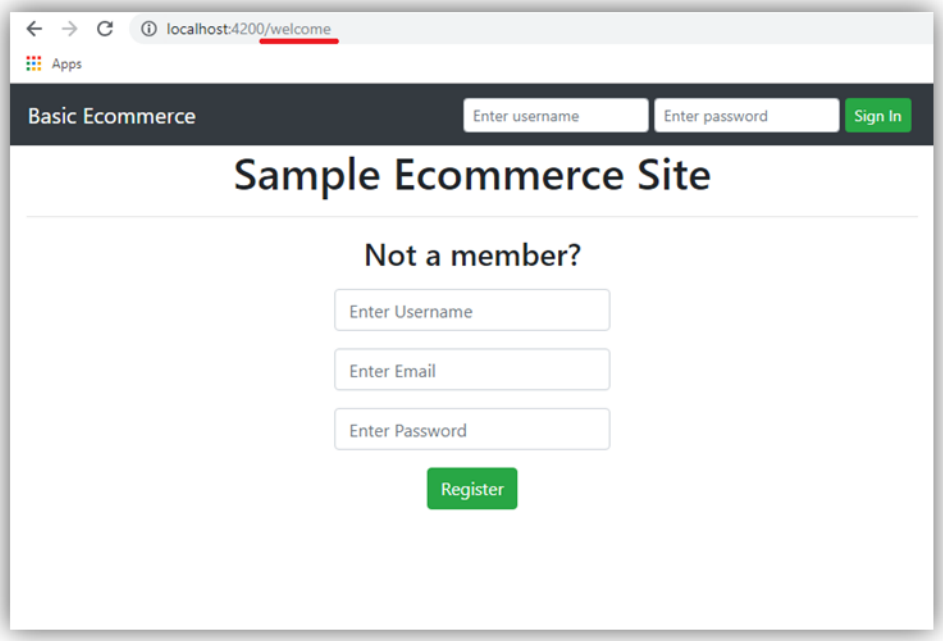
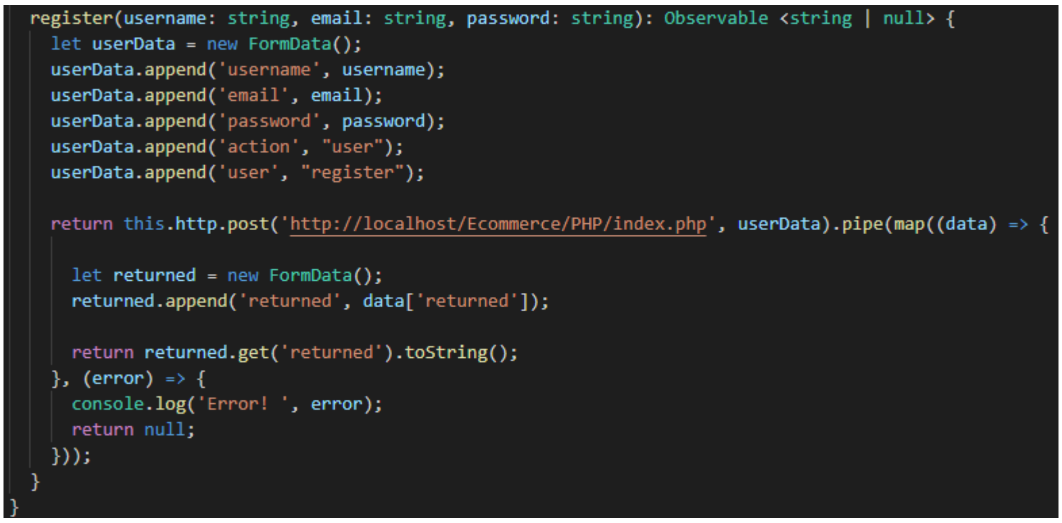


Figure 32 Welcome component

The welcome component allows customers to register. Once the input fields have been filled with data, the register button is clicked which triggers the register function in the welcome components' typescript file. This function sends the contents of the input fields to the HTTP service class to allow the customer to register with the site.

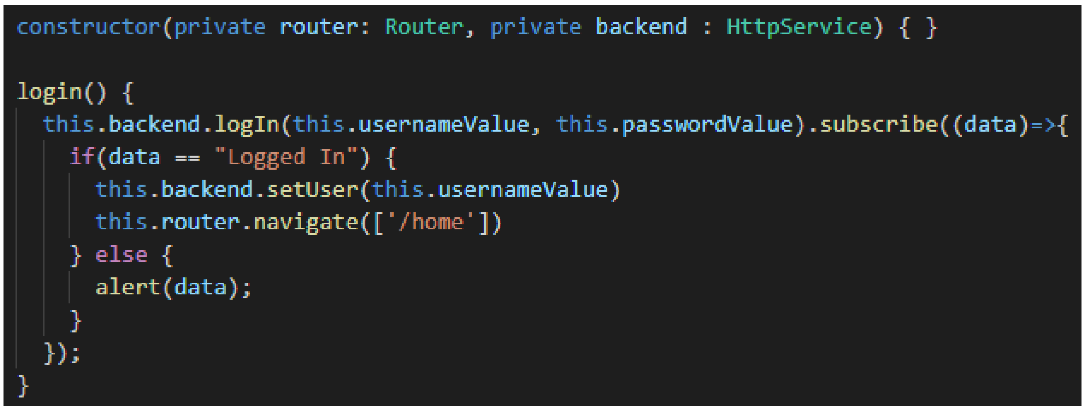


```
  register(username: string, email: string, password: string): Observable <string | null> {
    let userData = new FormData();
    userData.append('username', username);
    userData.append('email', email);
    userData.append('password', password);
    userData.append('action', "user");
    userData.append('user', "register");

    return this.http.post('http://localhost/Ecommerce/PHP/index.php', userData).pipe(map((data) => {

      let returned = new FormData();
      returned.append('returned', data['returned']);

      return returned.get('returned').toString();
    }, (error) => {
      console.log('Error! ', error);
      return null;
    }));
  }
}
```

Figure 33 Register function in HTTP Service class

The customers chosen email, username and password are packaged together with an action specifier and user marker before all are sent to the back end. The action tells the back end that it’s a message regarding the user (not for the negotiation system) and the user marker being set as ‘register’ tells the back end that the user wishes to register, not log-in. If the registration process is a failure, the error message (e.g. “Username taken”) is returned from the back end and alerted to the browser. If the registration is a success, the customer is logged in. If a log-in is detected, the welcome component uses the router to change the URL to ‘/home’ which signals to the router to update the display to the component associated with that URL, which is the home component.

If the customer has already registered with the site, the navbar can be used to log-in. Once the username and password have been entered, the sign in button can be clicked. This triggers the login function in the navbars typescript file.



```
constructor(private router: Router, private backend : HttpService) { }

login() {
  this.backend.logIn(this.usernameValue, this.passwordValue).subscribe((data)=>{
    if(data == "Logged In") {
      this.backend.setUser(this.usernameValue)
      this.router.navigate(['/home'])
    } else {
      alert(data);
    }
  });
}
```

Figure 34 Login function in navbar typescript class

The HTTP service and router are both injected into the navbar component through its constructor. The http services login function gets called which sends the entered username and password to the back end to be verified. As the backend needs to reply to the login request, subscribing to the login function prevents the login function from returning until the backend replies. This reply, if it contains the message "logged in", stores the username for future use and changes the URL to '/home'. This update to the URL prompts the router to display the home component.

4.2.4 Home Component

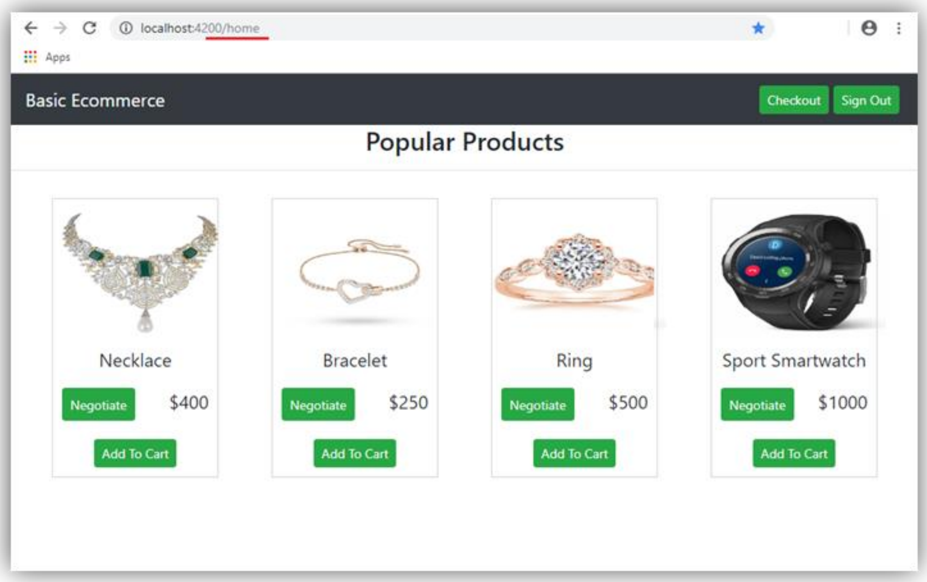


Figure 35 Home Component

The one navbar component is used in both the home and welcome component; it changes which buttons appear based on where it is being viewed. The sign in inputs and button are hidden once the customer logs in, with check out and sign out buttons appearing in their place.

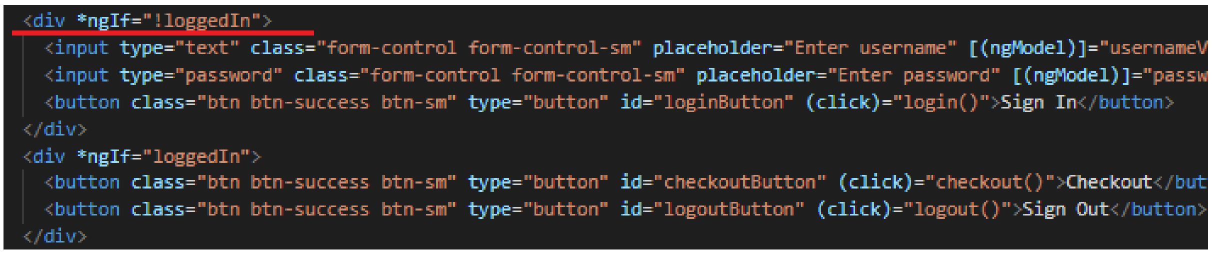
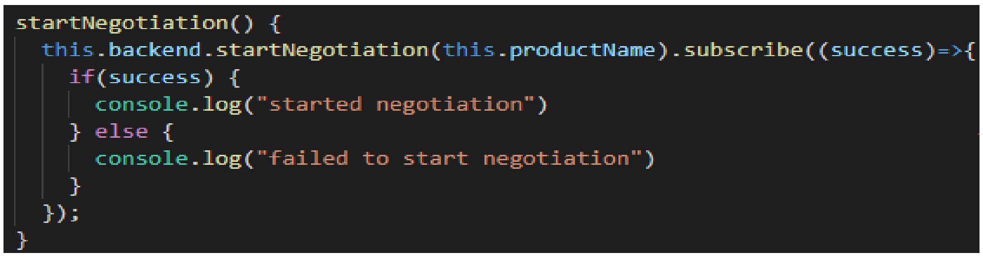

```
<div *ngIf="!loggedIn">
  <input type="text" class="form-control form-control-sm" placeholder="Enter username" [(ngModel)]="usernameV
  <input type="password" class="form-control form-control-sm" placeholder="Enter password" [(ngModel)]="passw
  <button class="btn btn-success btn-sm" type="button" id="loginButton" (click)="login()">Sign In</button>
</div>
<div *ngIf="loggedIn">
  <button class="btn btn-success btn-sm" type="button" id="checkoutButton" (click)="checkout()">Checkout</but
  <button class="btn btn-success btn-sm" type="button" id="logoutButton" (click)="logout()">Sign Out</button>
</div>
```

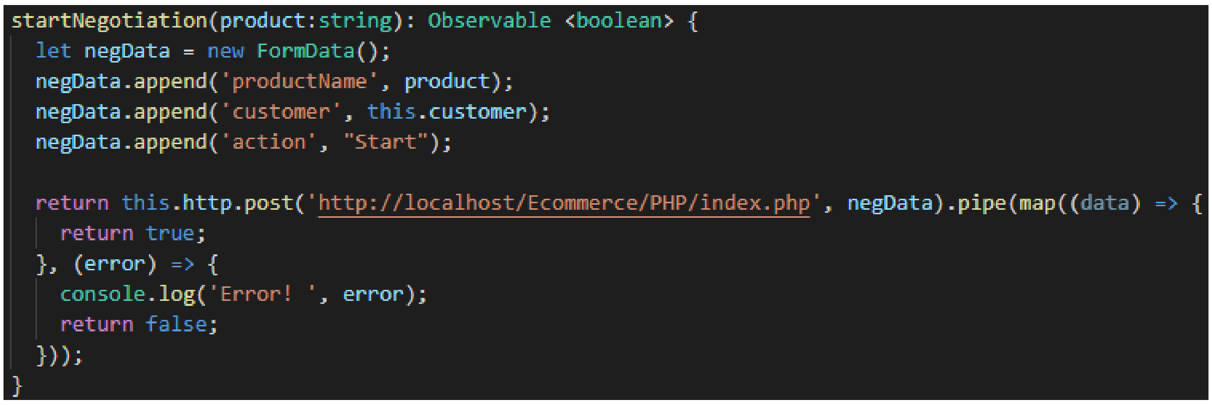
Figure 36 Navbar changes view based on whether the customer is logged in

Customers can now select a product on the home page to negotiate on by clicking the negotiate button. This triggers a function that uses the http service to notify the backend to start a negotiation.



```
startNegotiation() {
  this.backend.startNegotiation(this.productName).subscribe((success)=>{
    if(success) {
      console.log("started negotiation")
    } else {
      console.log("failed to start negotiation")
    }
  });
}
```

Figure 37 Home component telling the backend (via HTTP service class) to start the negotiation



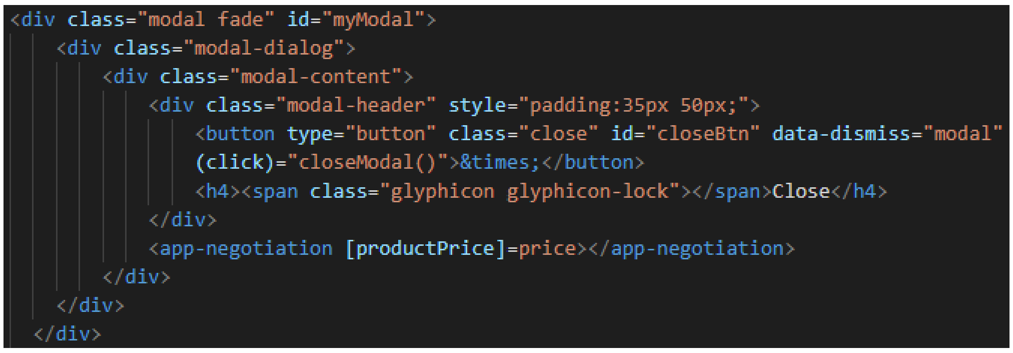
```
startNegotiation(product:string): Observable <boolean> {
  let negData = new FormData();
  negData.append('productName', product);
  negData.append('customer', this.customer);
  negData.append('action', "Start");

  return this.http.post('http://localhost/Ecommerce/PHP/index.php', negData).pipe(map((data) => {
    return true;
  }, (error) => {
    console.log('Error! ', error);
    return false;
  }));
}
```

Figure 38 StartNegotiation function in HTTP service class

The HTTP service class (‘backend’ in home’s typescript class) takes in a product name and sends this to the back end with a piece of ‘action’ data with “start” associated with it. Once the back end reads the action and sees “start”, it will know that the customer has selected a product and wishes to begin a negotiation on its price. The product name, and the customer’s name are also both sent to the backend, so that the negotiation system can know who its negotiating with and on what.

4.2.5 Negotiation Component

Once the negotiation starts, the negotiation modal will open. The home components modal fills itself with the negotiation component, passing it the price of the product for initial display purposes.



```
<div class="modal fade" id="myModal">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header" style="padding:35px 50px;">
                <button type="button" class="close" id="closeBtn" data-dismiss="modal"
                (click)="closeModal()">&times;</button>
                <h4><span class="glyphicon glyphicon-lock"></span>Close</h4>
            </div>
            <app-negotiation [productPrice]=price></app-negotiation>
        </div>
    </div>
  </div>
```

Figure 39 Home components Modal containing negotiation component

When the negotiation modal opens, the product price (the 'store owners' initial offer) is displayed on the right. The customer can then use the slider to set their offer, before sending it with the Offer button. If the customer wishes to send an offer that represents the most they are willing to pay for the product, the offer can be sent as the customers' final offer, by clicking the final offer button.

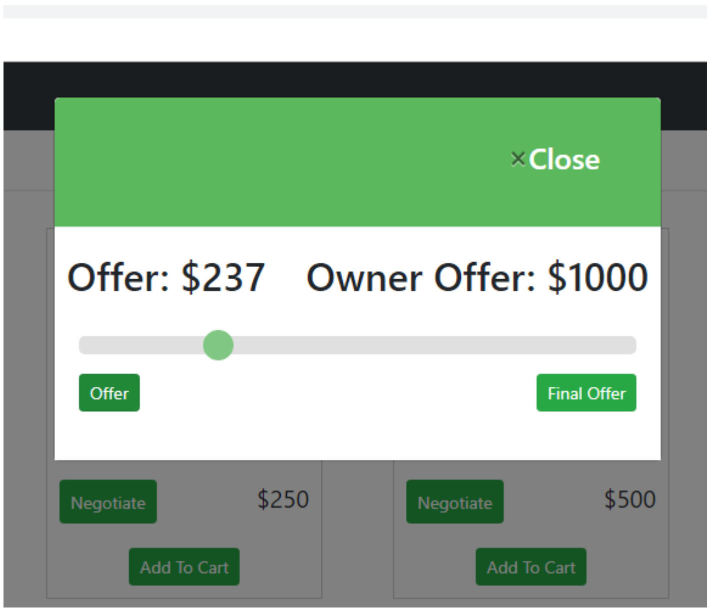
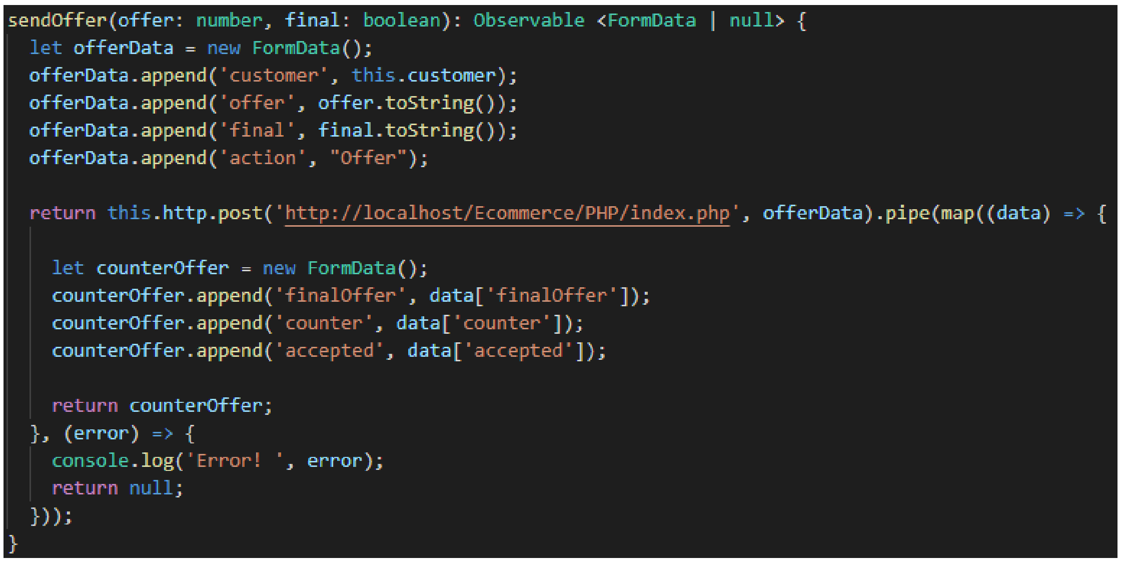


Figure 40 Offering 237 for product in modal containing the negotiation component

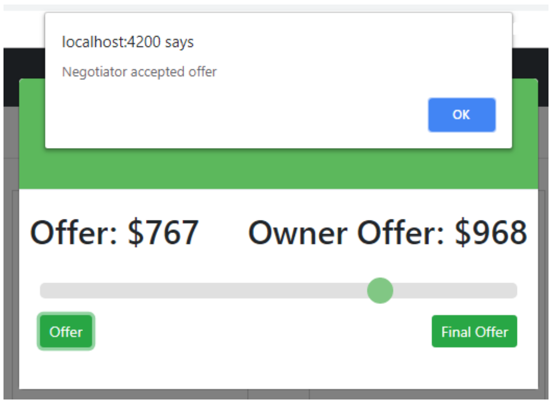
Once the offer or final offer buttons are pressed, the value of the offer is sent to the http service class to be sent to the back end.



```
sendOffer(offer: number, final: boolean): Observable <FormData | null> {
  let offerData = new FormData();
  offerData.append('customer', this.customer);
  offerData.append('offer', offer.toString());
  offerData.append('final', final.toString());
  offerData.append('action', "Offer");

  return this.http.post('http://localhost/Ecommerce/PHP/index.php', offerData).pipe(map((data) => {

    let counterOffer = new FormData();
    counterOffer.append('finalOffer', data['finalOffer']);
    counterOffer.append('counter', data['counter']);
    counterOffer.append('accepted', data['accepted']);

    return counterOffer;
  }, (error) => {
    console.log('Error! ', error);
    return null;
  }));
}
```

Figure 41 HTTP services SendOffer function used to send customer offers to the negotiation system

The customer’s name is sent so the negotiation system knows who the offer is from, and a string to indicate if the offer from the customer is their final offer. The action is ‘offer’, so the negotiation system knows what it just received was in fact, an offer. The negotiation system will reply to this offer with either a counter offer (which may or may not be final) or a true/false ‘accepted’ variable, true indicating the negotiation system accepted their offer, false indicating they rejected their offer (if the offer was final). The accepted variable can be empty, which indicates the customers offer want rejected or accepted, instead a counter offer was sent. This data is sent back to the negotiation component to process. If a counter offer is sent back, the slider is reset, the minimum offer the customer can now send is the previous sent offer + 1 and the maximum offer they can send is set equal to the just received counter offer. If the negotiation system sent back an accept or reject to the offer the customer is notified which via an alert, and the negotiation is ended. If the negotiation system sent back a counter and indicated it to be the final counter offer, the customer is given the option to accept or reject it. Either option ends the negotiation. Negotiations are ended via the HTTP services EndNegotiation method.


localhost:4200 says

Negotiator accepted offer

Figure 42 Negotiation system accepted the offer the customer sent

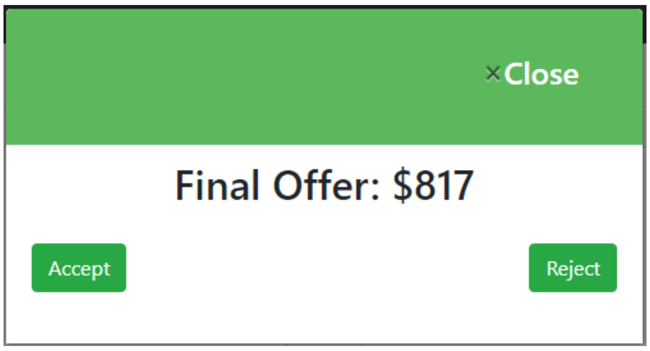


Figure 43 Negotiation system returned a counter set as final, customer must accept or reject it

4.3 Back-End Development

The back-end implementation involves creating a user server and negotiation agent to handle the user and negotiation requests that will be received from the front-end eCommerce site and negotiation component.

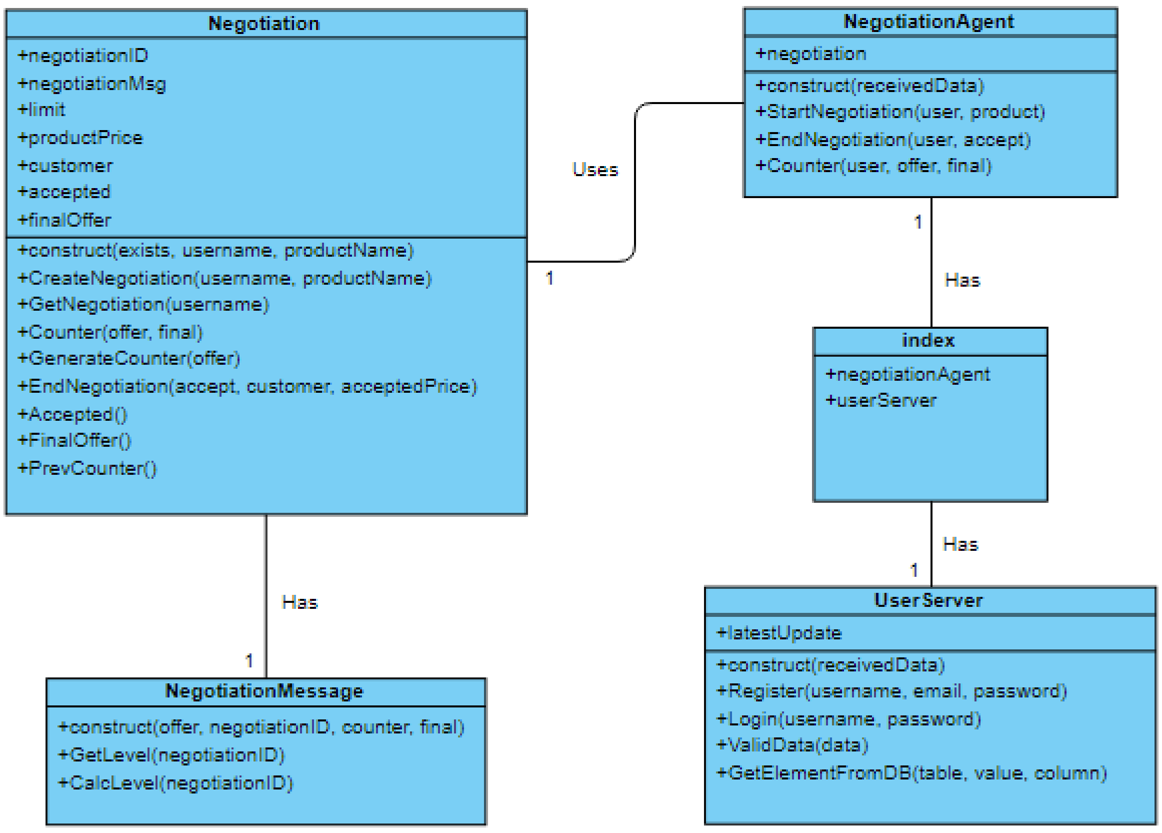
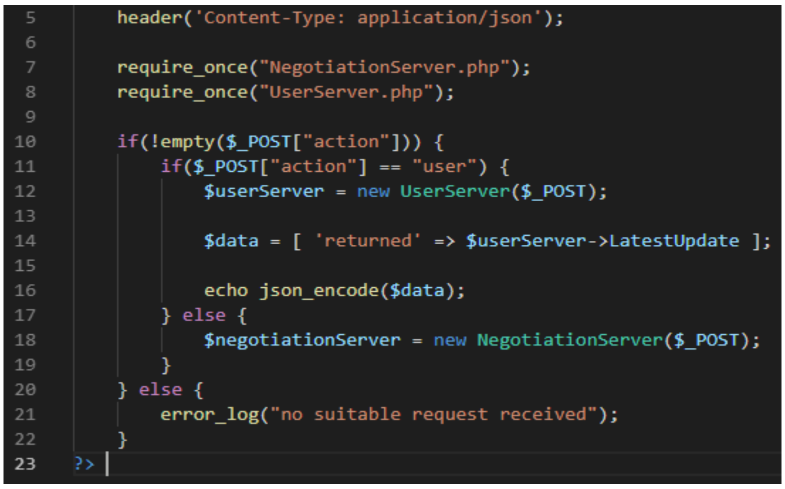

Negotiation
+negotiationID
+negotiationMsg
+limit
+productPrice
+customer
+accepted
+finalOffer
+construct(exists, username, productName)
+CreateNegotiation(username, productName)
+GetNegotiation(username)
+Counter(offer, final)
+GenerateCounter(offer)
+EndNegotiation(accept, customer, acceptedPrice)
+Accepted()
+FinalOffer()
+PrevCounter()
Uses
1
NegotiationAgent
+negotiation
+construct(receivedData)
+StartNegotiation(user, product)
+EndNegotiation(user, accept)
+Counter(user, offer, final)
1
Has
index
+negotiationAgent
+userServer
Has
1
UserServer
+latestUpdate
+construct(receivedData)
+Register(username, email, password)
+Login(username, password)
+ValidData(data)
+GetElementFromDB(table, value, column)
Has
1
NegotiationMessage
+construct(offer, negotiationID, counter, final)
+GetLevel(negotiationID)
+CalcLevel(negotiationID)

Figure 44 Initial class diagram for negotiation system

4.3.1 Index Class

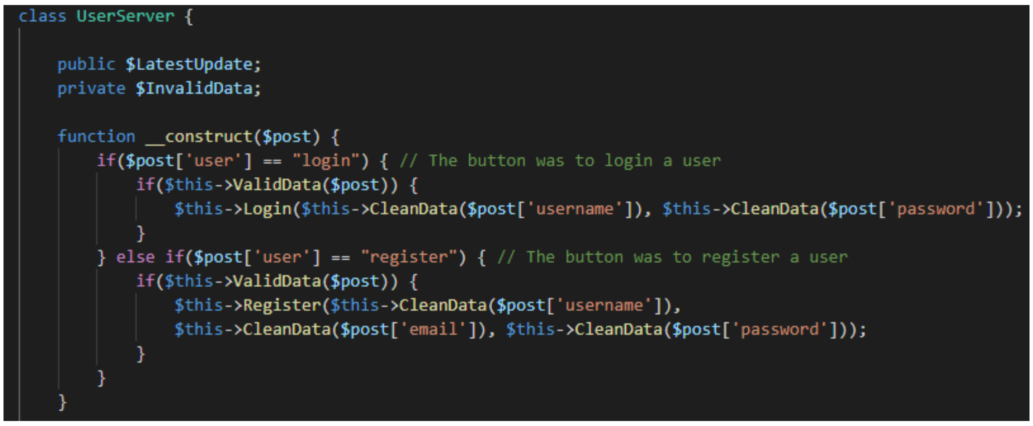
All code for the back end was created inside Visual Code. The PHP classes were stored inside a PHP folder inside the main ‘ecommerce’ angular app folder. To begin, an index.php class was created. This class received the JSON messages from the front-end. The messages were differentiated with an ‘action’ field. If the field was equal to user, then the message contents were sent to the UserServer class. If not equal to user, the message contents were meant for and sent to the NegotiationAgent. The UserServer was given a variable to hold any updates that need to be sent back to the front-end. As such, once index.php sends the message to UserServer, it grabs the latest update, wraps it in JSON format and echo’s it back to the front-end.



```php
    header('Content-Type: application/json');

    require_once("NegotiationServer.php");
    require_once("UserServer.php");

    if(!empty($_POST["action"])) {
        if($_POST["action"] == "user") {
            $userServer = new UserServer($_POST);

            $data = [ 'returned' => $userServer->LatestUpdate ];

            echo json_encode($data);
        } else {
            $negotiationServer = new NegotiationServer($_POST);
        }
    } else {
        error_log("no suitable request received");
    }
?>
```

Figure 45 index.php class

4.3.2 User Server



```php
class UserServer {

    public $LatestUpdate;
    private $InvalidData;

    function __construct($post) {
        if($post['user'] == "login") { // The button was to login a user
            if($this->ValidData($post)) {
                $this->Login($this->CleanData($post['username']), $this->CleanData($post['password']));
            }
        } else if($post['user'] == "register") { // The button was to register a user
            if($this->ValidData($post)) {
                $this->Register($this->CleanData($post['username']),
                $this->CleanData($post['email']), $this->CleanData($post['password']));
            }
        }
    }
```

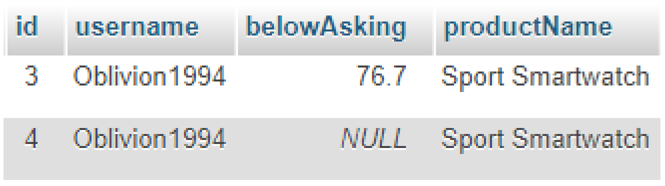
Figure 46 UserServer constructor

UserServer takes in the data that the index passes it. UserServer check the ‘user’ field to see whether the request is to login or register. The class then uses the ValidData() method to verify the credentials are both present and in the correct format. If so, the class will either log-in or register the user, passing in cleaned versions of the credentials (special characters stripped, slashes stripped) to the respective function. The Register and Login functions attempt to do both and update the latestUpdate variable if it’s a success, or failure (and the reason why).

If the request sent to index.php does not have the ‘action’ field set to user, then the request is for the negotiation agent.

4.3.3 Negotiation Agent

The negotiation agent takes in the request and reads another field called ‘action’. Action can be 1 of 3 possible values; “Start”, “Offer” or “End”. “Start” and “End” either start or end a negotiation. Start calls the StartNegotiation function, passing in the username for the customer who wishes to start negotiating, and the name of the product they wish to negotiate on. This results in a negotiation row being entered into the negotiations table in the database, with the negotiation being assigned a unique id and the username/product name being stored.

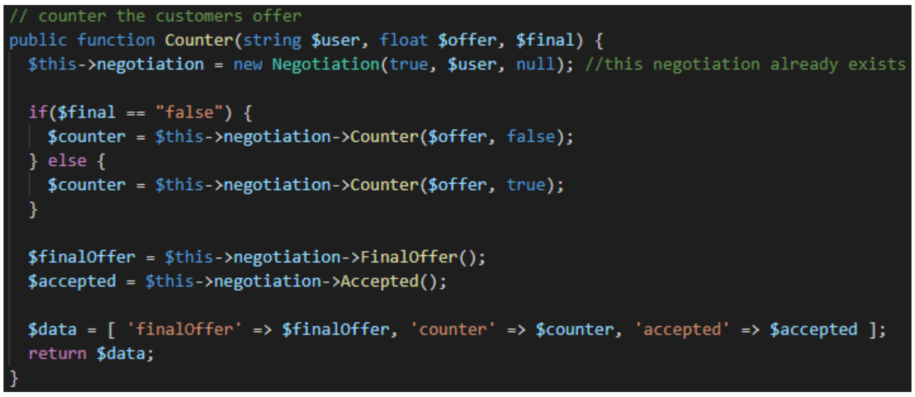


id	username	belowAsking	productName
3	Oblivion1994	76.7	Sport Smartwatch
4	Oblivion1994	*NULL*	Sport Smartwatch

Figure 47 Sample entries in the negotiations table

“End” results in the EndNegotiation function being called, the username of the customer who sent the request gets passed in, so the system knows which negotiation to end.

“Offer” triggers the Counter function. This function is used when a customer wishes to make an offer on a product and so sends it to the negotiation system. The function takes in 3 variables; the username of the customer, their offer and a string indicating whether their offer is final or not.



```
// counter the customers offer
public function Counter(string $user, float $offer, $final) {
  $this->negotiation = new Negotiation(true, $user, null); //this negotiation already exists

  if($final == "false") {
    $counter = $this->negotiation->Counter($offer, false);
  } else {
    $counter = $this->negotiation->Counter($offer, true);
  }

  $finalOffer = $this->negotiation->FinalOffer();
  $accepted = $this->negotiation->Accepted();

  $data = [ 'finalOffer' => $finalOffer, 'counter' => $counter, 'accepted' => $accepted ];
  return $data;
}
```

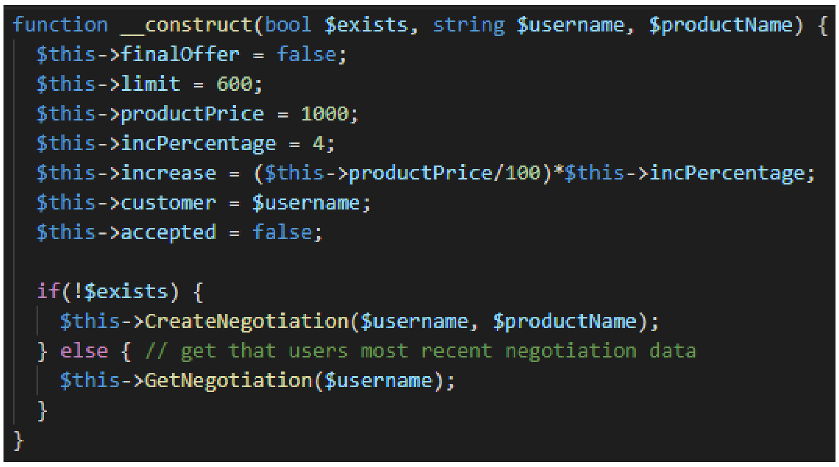
Figure 48 Counter method for NegotiationAgent class

Like with the start and end negotiation function, the first thing that occurs is the instantiation of a negotiation class. The function indicates to this class that the negotiation already exists (as you must have already started a negotiation to send an offer) via 'true', and the username of the customer who is negotiating. The negotiation has already been created, so no product name needs to be sent to the negotiation class (hence the null). The function then calls the negotiation counter method, passing in the customers offer and whether it is their final offer or not. The negotiation systems counter to this offer is then returned in response to the customers offer. The function checks to see if the counter is the negotiating systems final counter, or if the system accepted the customers offer or not. This information is saved and returned to the index class to be sent back to the customer on the front-end.

Regardless of whether a user wishes to start a negotiation, end a negotiation or send an offer for an existing negotiation, a negotiation class instance is always created.

4.3.4 Negotiation Class

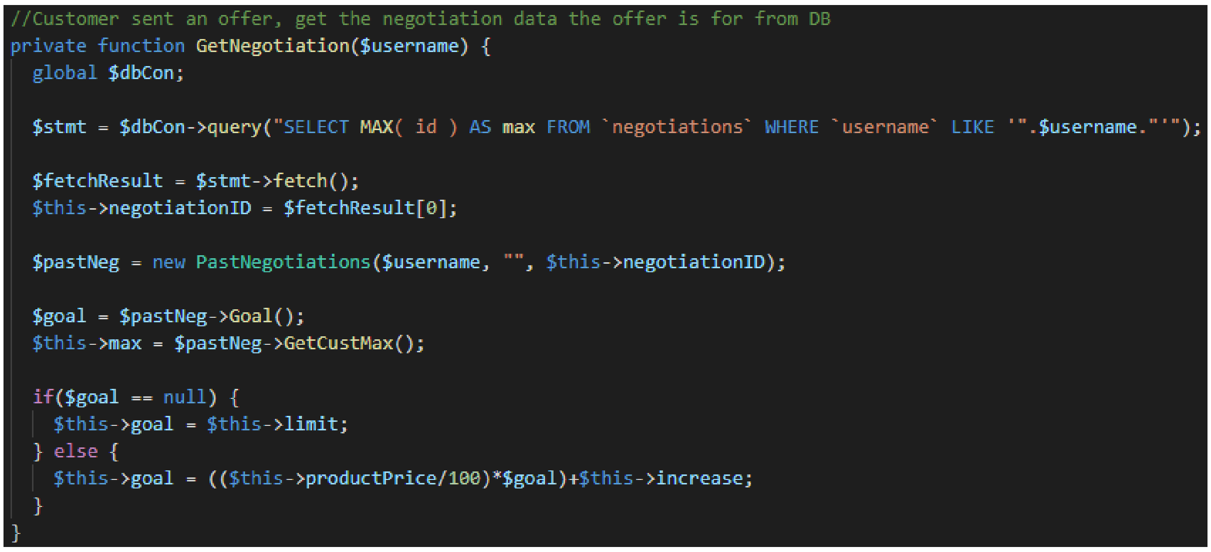
The negotiation class is the main class of the system. It is used by the negotiation agent to start or end negotiations and counter any offers sent its way.



```php
function __construct(bool $exists, string $username, $productName) {
  $this->finalOffer = false;
  $this->limit = 600;
  $this->productPrice = 1000;
  $this->incPercentage = 4;
  $this->increase = ($this->productPrice/100)*$this->incPercentage;
  $this->customer = $username;
  $this->accepted = false;

  if(!$exists) {
    $this->CreateNegotiation($username, $productName);
  } else { // get that users most recent negotiation data
    $this->GetNegotiation($username);
  }
}
```

Figure 49 Negotiation class constructor

If the negotiation has already started, then the GetNegotiation method is called which returns the negotiation id for the negotiation the customer is currently engaged in. If not, the CreateNegotiation method is called so a negotiation entry is added to the database for the customer and the product they are negotiating on, which in turn gets a unique negotiation id assigned to the negotiation to allow future updates to update the correct negotiation. The limit for the system is hardcoded here, this represents the lowest price the negotiation agent can let the product go for. The increment percentage variable is set to 4, which is used to calculate what the increase should be to the negotiation agents price goal following each successful negotiation. The 'goal' for the current negotiation gets calculated in the GetNegotiation method.



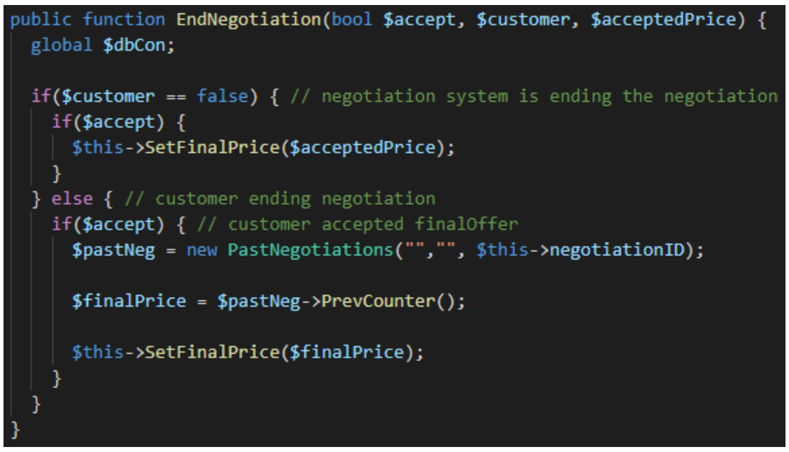
```php
//Customer sent an offer, get the negotiation data the offer is for from DB
private function GetNegotiation($username) {
  global $dbCon;

  $stmt = $dbCon->query("SELECT MAX( id ) AS max FROM `negotiations` WHERE `username` LIKE '".$username."'");

  $fetchResult = $stmt->fetch();
  $this->negotiationID = $fetchResult[0];

  $pastNeg = new PastNegotiations($username, "", $this->negotiationID);

  $goal = $pastNeg->Goal();
  $this->max = $pastNeg->GetCustMax();

  if($goal == null) {
    $this->goal = $this->limit;
  } else {
    $this->goal = (($this->productPrice/100)*$goal)+$this->increase;
  }
}
```

Figure 50 Negotiation class GetNegotiation method

The PastNegotiations class can be used to get past negotiation data about the customer. Using this, the goal is retrieved. The goal represents the last successful negotiation the system reached with the customer, if any. This value is represented as a % below the original asking price. If none exist, the negotiation agents limit is set as the goal for the current negotiation. If one does exist, the goal is calculated for the current product price and an increase is added on, so to ensure the goal of the system is higher than the previous agreed upon price. The past negotiation class is also used to retrieve the customers maximum offer, if it is known. If the max offer the customer is known to go to is known, the system will just offer that as the final price to the customer, if it is above the limit. Asking the customer for more than the max could result in a rejection from the customer, and asking for less than the max could result on a settlement price lower than the system could have gotten.

The negotiation class was also created to be used by the negotiation agent to calculate the appropriate counter for a given offer. A counter function in the negotiation class is called by the negotiation agent for this very purpose. The function takes in the customers offer and a bool representing if the customers offer was their final offer or not. The counter function makes the following decisions in order:

1. If the offer is the customers final offer, record it, so it can be known in future negotiations that this offer is the most a customer will pay during a negotiation.
2. If the offer is final, check to see if the offer is greater than or equal to the negotiation systems limit. In the case of final offers, the limit takes precedence over the goal. At this point, since the offer is final, we know this is the most the customer will pay (i.e. their limit). There is no point in the system using the goal value (previous negotiation agreed amount + increment) to decide whether to accept the customers final offer if the customers limit. If the customers limit is below the systems goal but above the systems limit, by right a negotiation could have been reached but by using the goal, the negotiation was ended without an agreement. The system should always prioritise reaching an agreement over reaching a better agreement than last time.
3. If the offer is final and is above or equal to the systems limit, accept the offer, and store it as a message in the negotiation. At this point, end the negotiation.
4. If the offer is final but is below the systems limit, reject the offer, and store it as a message in the negotiation. At this point, end the negotiation.

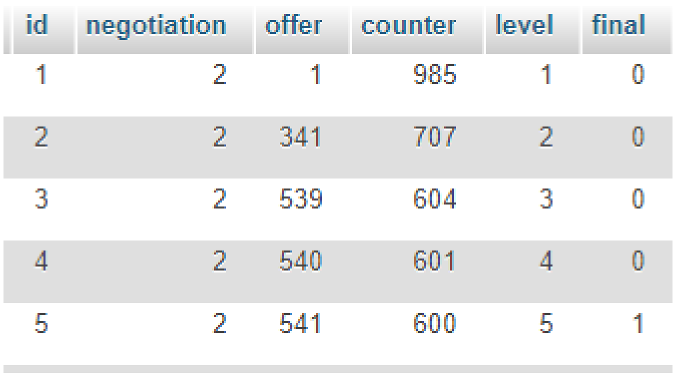


```
public function EndNegotiation(bool $accept, $customer, $acceptedPrice) {
  global $dbCon;

  if($customer == false) { // negotiation system is ending the negotiation
    if($accept) {
      $this->SetFinalPrice($acceptedPrice);
    }
  } else { // customer ending negotiation
    if($accept) { // customer accepted finalOffer
      $pastNeg = new PastNegotiations("","", $this->negotiationID);

      $finalPrice = $pastNeg->PrevCounter();

      $this->SetFinalPrice($finalPrice);
    }
  }
}
```

Figure 51 EndNegotition method in Negotiation class

Ending a negotiation can be done by either the negotiation agent or the customer themselves. The customer variable is a true false value indicating if the customer is the one ending the negotiation or not. If not, it's the agent, and the final price agreed on with the customer gets recorded for future negotiations (only if the agent accepted the final price). Likewise, if the customer ends the negotiation the final price they agreed on needs to be recorded for future negotiations (again, only if the price was accepted). Negotiations ending with one side rejecting the others final offer do not have their resulting price recorded. Due to the stateless nature of PHP, by time the customer accepts or rejects the agents offer on the front-end, this offer amount is already lost. PastNegotiations must be used to get the last counter sent by the negotiation agent, as this will have been the price accepted or rejected by the customer.

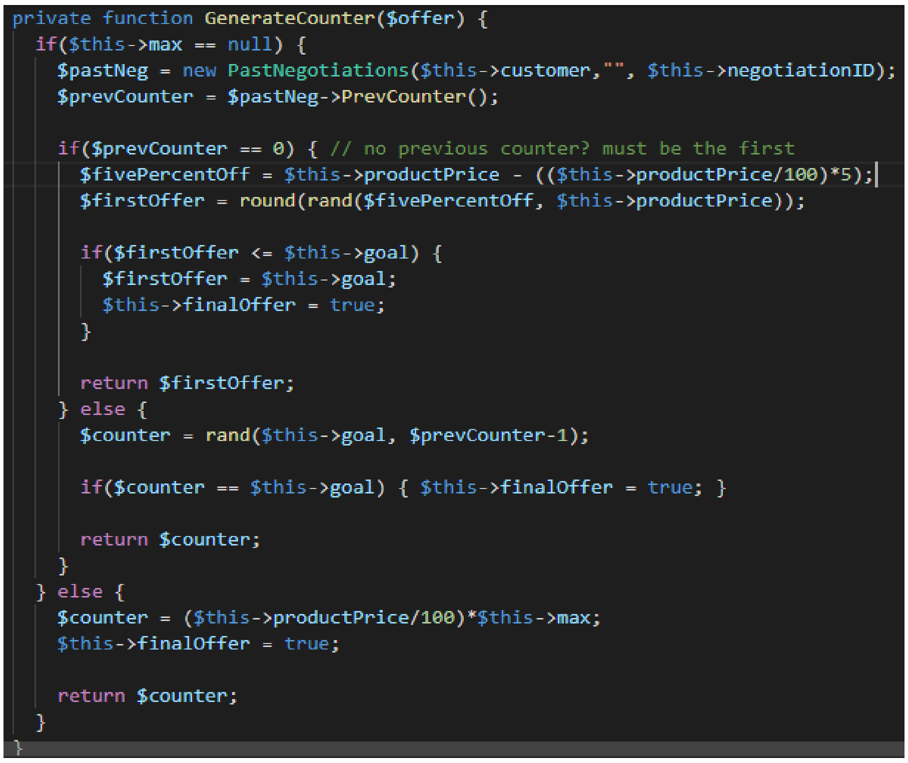
5. If the offer sent by the customer is not final, the system checks to see if it is greater than or equal to the systems goal. If so, the offer is accepted, the offer is stored as a message in the negotiation and the negotiation is ended.



id	negotiation	offer	counter	level	final
1	2	1	985	1	0
2	2	341	707	2	0
3	2	539	604	3	0
4	2	540	601	4	0
5	2	541	600	5	1

Figure 52 All messages stored in a database for a negotiation of id 2

6. If the offer is less than the goal of the system, the system generates a counter via the GenerateCounter function, stores it as message in the negotiation and returns it to index.php to be sent to the customer.



```php
private function GenerateCounter($offer) {
  if($this->max == null) {
    $pastNeg = new PastNegotiations($this->customer,"", $this->negotiationID);
    $prevCounter = $pastNeg->PrevCounter();

    if($prevCounter == 0) { // no previous counter? must be the first
      $fivePercentOff = $this->productPrice - (($this->productPrice/100)*5);
      $firstOffer = round(rand($fivePercentOff, $this->productPrice));

      if($firstOffer <= $this->goal) {
        $firstOffer = $this->goal;
        $this->finalOffer = true;
      }

      return $firstOffer;
    } else {
      $counter = rand($this->goal, $prevCounter-1);

      if($counter == $this->goal) { $this->finalOffer = true; }

      return $counter;
    }
  } else {
    $counter = ($this->productPrice/100)*$this->max;
    $this->finalOffer = true;

    return $counter;
  }
}
```

Figure 53 Generate Counter method in Negotiation class

Counter generation in the negotiation system involves making the following decisions:

1. If the max percentage of the product price a customer is willing to pay during negotiations is known, the counter is set to this, and returned as the final offer by the negotiation system.
2. If the max is not known (is null), the PastNegotiations class is instantiated and used to get the previous counter sent by the system in the current negotiation.
3. If it turns out this is the first offer the system has received by the customer in the current negotiation (previous counter doesn't exist), the first offer is generated, set as a maximum of 5 percent below starting price, and returned.
4. If a previous counter in the negotiation exists, the next counter is selected as a number between the previous counter minus 1 and the systems goal.

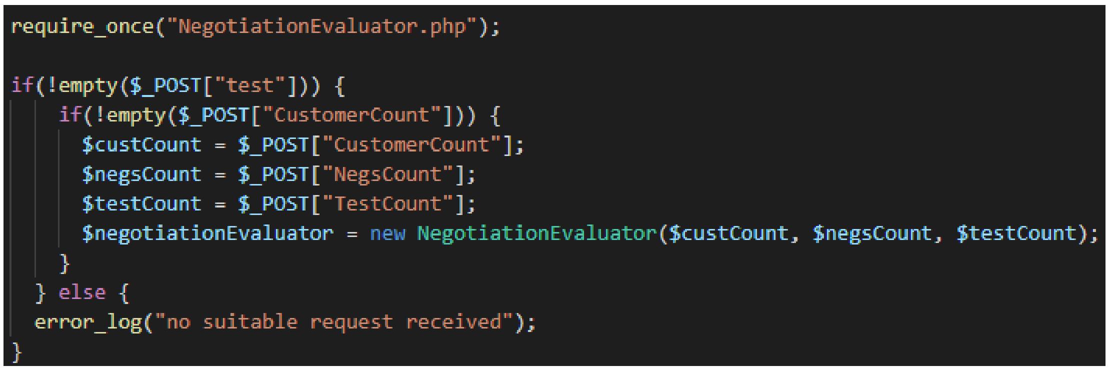
5. If the generated counter is equal to the systems goal for the negotiation, the counter is set as the final before being returned. As at this point, the system can't offer anything lower or it won't reach its goal.

4.4 Negotiation Evaluator System

4.4.1 Index Class

The negotiation evaluator system was created to both test the negotiation system during the development process and as an aid in the evaluation of the negotiation system after development is completed. A simple Angular component was created with three input fields. One field allowed the number of customers to be set, another field sets how many negotiations each customer is made perform and the final field sets the amount of the times the evaluator is ran.

The HTTP service class sends these values to an index.php class to start the evaluator.

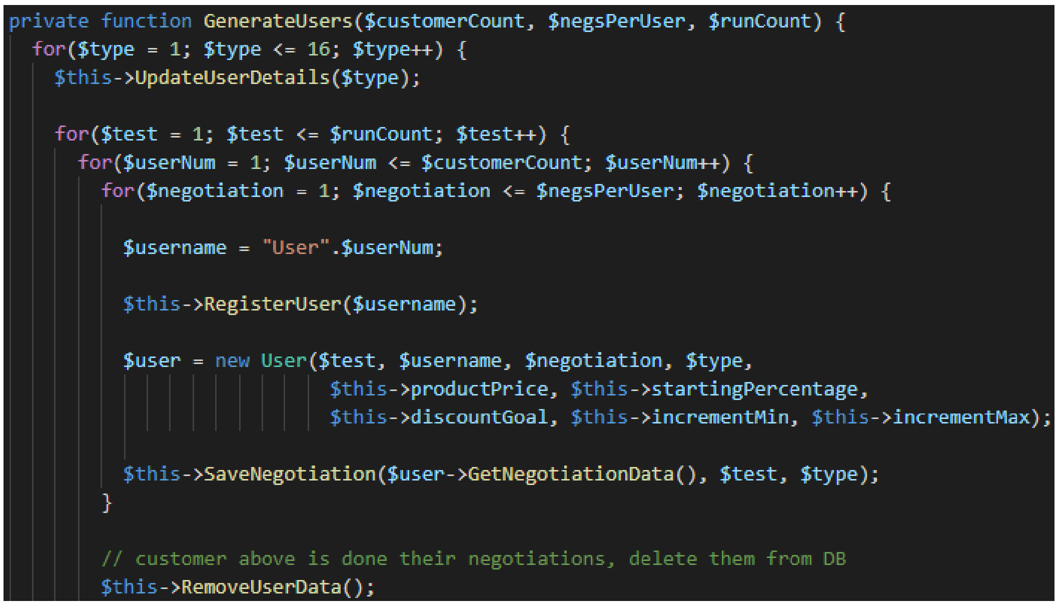


```
require_once("NegotiationEvaluator.php");

if(!empty($_POST["test"])) {
    if(!empty($_POST["CustomerCount"])) {
      $custCount = $_POST["CustomerCount"];
      $negsCount = $_POST["NegsCount"];
      $testCount = $_POST["TestCount"];
      $negotiationEvaluator = new NegotiationEvaluator($custCount, $negsCount, $testCount);
    }
  } else {
  error_log("no suitable request received");
}
```

Figure 54 index.php receives the field values from the front-end and creates the evaluator with them

4.4.2 Negotiation Evaluator Class

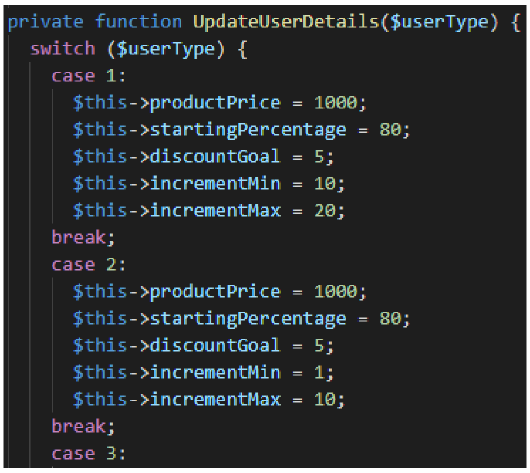
Once the NegotiationEvaluator is instantiated, it begins to run.



```
private function GenerateUsers($customerCount, $negsPerUser, $runCount) {
  for($type = 1; $type <= 16; $type++) {
    $this->UpdateUserDetails($type);

    for($test = 1; $test <= $runCount; $test++) {
      for($userNum = 1; $userNum <= $customerCount; $userNum++) {
        for($negotiation = 1; $negotiation <= $negsPerUser; $negotiation++) {

          $username = "User".$userNum;

          $this->RegisterUser($username);

          $user = new User($test, $username, $negotiation, $type,
                           $this->productPrice, $this->startingPercentage,
                           $this->discountGoal, $this->incrementMin, $this->incrementMax);

          $this->SaveNegotiation($user->GetNegotiationData(), $test, $type);
        }

        // customer above is done their negotiations, delete them from DB
        $this->RemoveUserData();
```

Figure 55 Evaluators function for generating customers to negotiate with the system

The NegotiatonEvaluator constructor calls the GenerateUsers function. Here, the customers behaviour variables are updated to whichever the current customer type the evaluator is creating.



```
private function UpdateUserDetails($userType) {
  switch ($userType) {
    case 1:
      $this->productPrice = 1000;
      $this->startingPercentage = 80;
      $this->discountGoal = 5;
      $this->incrementMin = 10;
      $this->incrementMax = 20;
    break;
    case 2:
      $this->productPrice = 1000;
      $this->startingPercentage = 80;
      $this->discountGoal = 5;
      $this->incrementMin = 1;
      $this->incrementMax = 10;
    break;
    case 3:
```

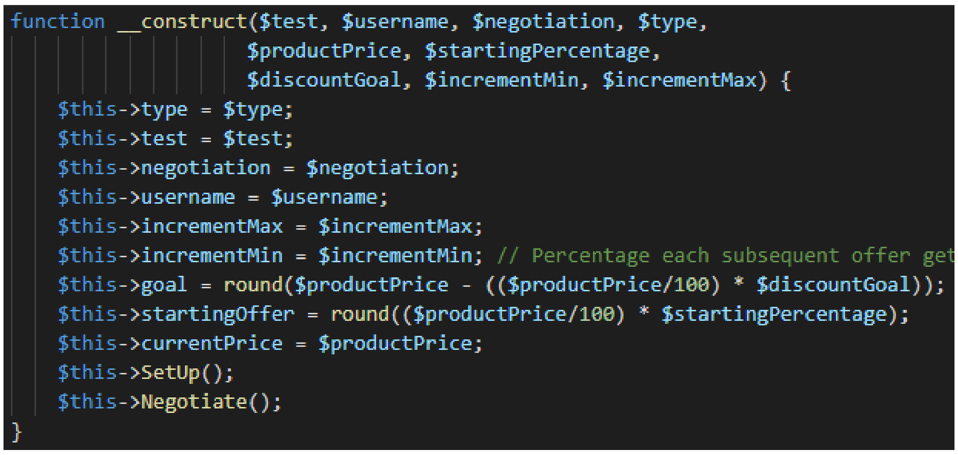
Figure 56 As the ‘userType’ variable increments, the customers negotiation behaviour changes

The evaluator will, for each of the 16 customer types, run a set number of times for each customer type (decided by ‘runCount’ value). The evaluator will create a set number of customers for each of the types (decided by ‘customerCount’ value). The evaluator will have each of these customers negotiate a set number of times (decided by the ‘negsPerUser’ value). For example, a ‘runCount’ of 5 with a ‘customerCount’ and ‘negsPerUser’ of 4 and 3 respectively, will result in 3 negotiations for each of the 4 customers for each of the 16 types. This will be repeated 5 times for a total of 960 negotiations (3 negotiations x 4 customers x 16 types x 5 runs).

4.4.3 User Class

When a user is created, it is given the negotiation behaviour set by the current type and is registered with the negotiation system (is added to the database like when a real customer registers). The user class is instantiated and negotiates with the negotiation system. The result of the negotiation result is saved to a file, and the user is removed from the database.

The User class negotiates with the negotiation system in a way specified by the behaviour give to it.



```
function __construct($test, $username, $negotiation, $type,
                     $productPrice, $startingPercentage,
                     $discountGoal, $incrementMin, $incrementMax) {
    $this->type = $type;
    $this->test = $test;
    $this->negotiation = $negotiation;
    $this->username = $username;
    $this->incrementMax = $incrementMax;
    $this->incrementMin = $incrementMin; // Percentage each subsequent offer get
    $this->goal = round($productPrice - (($productPrice/100) * $discountGoal));
    $this->startingOffer = round(($productPrice/100) * $startingPercentage);
    $this->currentPrice = $productPrice;
    $this->SetUp();
    $this->Negotiate();
}
```

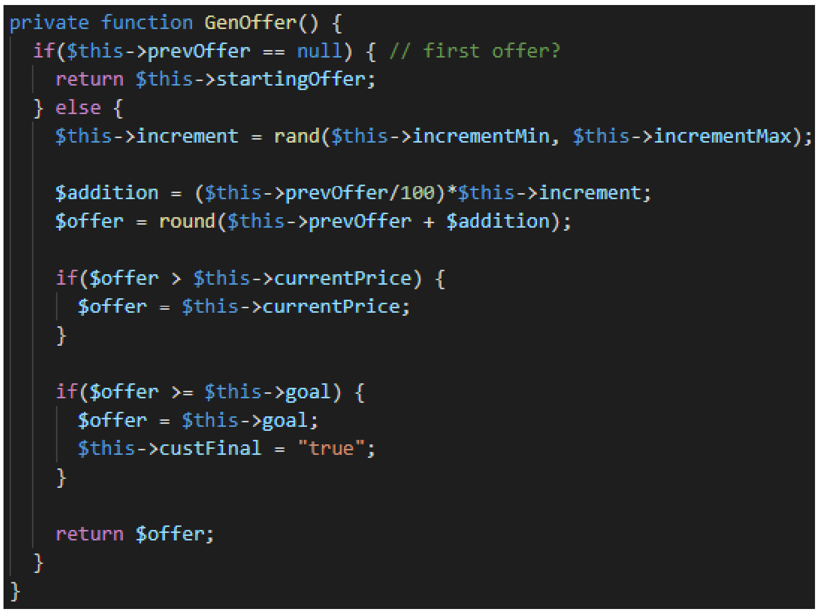
Figure 57 User constructor

The increment min and max specify the range which the user can generate its increments to its offers. The startingPercentage specifies the percentage of the products price the user will send as its first offer to the system and the discount goal is the price the user tries to reach in the negotiation. For example, a min of 20 and max of 30, a product price of 1000, a starting percentage of 50 and a discount goal of 30 are used.

This means the users first offer is 50% of 1000, any following offers will be the previous offer plus a random value between 20 and 30 and the user is trying to get a price 30% below the asking which is 700. Anything below 700 is not acceptable to the user.

Before starting the negotiation, the user has some basic set up. This includes initialising some variables and instantiating the negotiation agent. The user sends a message to the negotiation agent that it wishes to start the negotiation and so the Negotiate function is called to begin.

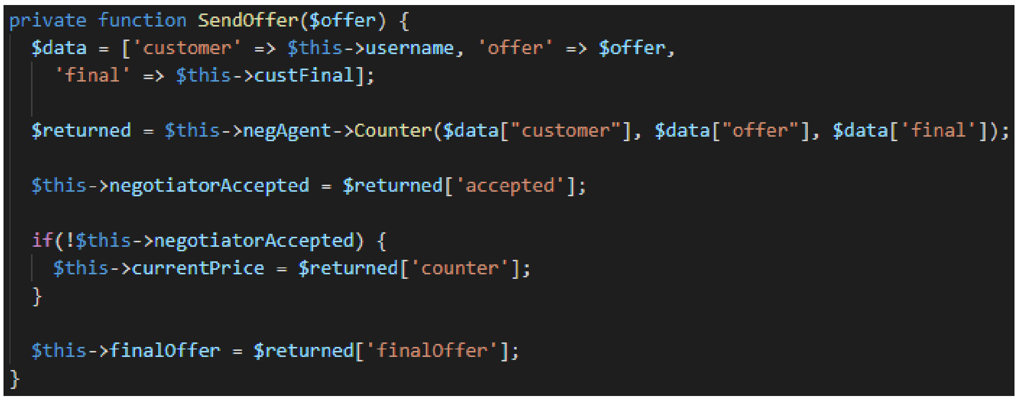
The user generates the first offer using its GenOffer function.



```php
private function GenOffer() {
  if($this->prevOffer == null) { // first offer?
    return $this->startingOffer;
  } else {
    $this->increment = rand($this->incrementMin, $this->incrementMax);

    $addition = ($this->prevOffer/100)*$this->increment;
    $offer = round($this->prevOffer + $addition);

    if($offer > $this->currentPrice) {
      $offer = $this->currentPrice;
    }

    if($offer >= $this->goal) {
      $offer = $this->goal;
      $this->custFinal = "true";
    }

    return $offer;
  }
}
```

Figure 58 GenOffer function

The GenOffer function checks to see if the user has sent an offer in the negotiation. If they haven’t, the starting offer is used as the current offer. If the user has sent an offer before, the increment is calculated as a percentage of the previous offer and added on (e.g. previous offer was 100, increment generated as 22, so 22% of 100 is added onto 100 to make the current offer). If the offer is greater than the current price of the product, it is fixed. If the offer goes over the goal of the user (e.g. max 700), the offer is set to the max and the user specifies that the offer is their final offer they are going to make. This offer is returned and sent to the negotiation agent via the SendOffer function.



```
private function SendOffer($offer) {
  $data = ['customer' => $this->username, 'offer' => $offer,
    'final' => $this->custFinal];

  $returned = $this->negAgent->Counter($data["customer"], $data["offer"], $data['final']);

  $this->negotiatorAccepted = $returned['accepted'];

  if(!$this->negotiatorAccepted) {
    $this->currentPrice = $returned['counter'];
  }

  $this->finalOffer = $returned['finalOffer'];
}
```

Figure 59 SendOffer function

The negotiation agent counter function is sent the users offer and the agents counter offer is saved. This counter offer, like it does for real customers, can contain counters, final counters, accepts or rejects. If the agent accepts or rejects the users final offer, the negotiation is ended, and the evaluator saves the result before moving on to the next user. If the user receives a counter offer, it will send another offer to try get a better price. If the counter was final and above the user's goal (above 700), the user will reject the negotiators final offer. If the agent's final offer is equal to or below the goal, the user will accept it.

4.5 *Experimental Implementation*

Two experiments were implemented. The main experiment, which focused on the store owner's satisfaction, was quantitative. The ancillary experiment, which focused on the customers satisfaction, was qualitative.

4.5.1 Store Owner Satisfaction Experiment

For the negotiation systems performance to be deemed satisfactory to a store owner, the negotiation system must obtain satisfactory results (achieve sales above limit) regardless of how many customers it must negotiate with, how many negotiations with each customer it must have and regardless of what its limit is set to. As such, the experiment consisted of 4 steps:

Step one: The negotiation evaluator was used to figure out what percentage increase to each subsequent negotiation, following a successful negotiation, generated the best results. This number needed to be figured out first, as the system needed to be set to use this number in the following 3 steps. The evaluator was used 7 times for this step, each time with one customer of each type negotiating 30 times. Each time a higher increase was set in the system than the previous. The increases were 1%, 2%, 3%, 4%, 5%, 10% and 20%. Step one was executed 10 times to get the average results for each increase.

Step two: Record the systems results when the number of customers the system must negotiate with is increased. The negotiation evaluator was used 6 times for this step, each time the customers of each type only had to negotiate 1 time each, but the number of customers was increased with each use. The step started with 1 customer, then 5, 10, 50, 100 and finally 250. Step two was executed 5 times to get the average results for all customer increases.

Step three: Record the systems results when the number of negotiations the system must engage in with each customer is increased. The negotiation evaluator was used 7 times for this step. Each time one customers of each type had to negotiate with the system, but the number of negotiations they had to do was increased with each use. The customers began having to do 1 negotiation, then this was increased to 5, then 10, 25, 50, 100 and finally 250. Step three was executed 5 times to get the average results for all negotiation increases.

Step four: Record the systems results when the limit the negotiation system must stick to when negotiating with each customer is increased. The negotiation evaluator was used 6 times for this step. Each time the limit was increased, starting with a limit below the max the customers were willing to pay and ending above it. 1 customer of each type had to negotiate with the system 10 times for each of the different limits. The price of the product remained the same at 1000 but the limit started at 500, and was then increased to 550, then 600, 700, 950 and finally 995. Step four was executed 5 times to get the average results for all negotiation increases.

4.5.2 Store Owner Satisfaction Experiment

5 volunteers were used to take part in this experiment. 5 people is considered enough in many situations for collecting user feedback (Nielson, 2000).

The volunteers were presented with the ecommerce stores welcome page and given limited instructions, which were as follows:

1. Register with the eCommerce store.
2. Choose a product.
3. Negotiate on its price.
4. Record the results of the negotiation. (Reached agreement? For how much?)
5. Repeat steps 2, 3 and 4 10 times.

The system's ability to learn each customer max price that they are willing to pay during negotiations was disabled. This feature wasn't designed to deal with customers who engage in dynamic negotiation strategies, and since the volunteers could employ any negotiation strategy, they saw fit to use, the feature had to be disabled. The feature to demand greater amounts with each successful negotiation with a customer remained enabled.

Part Two:

The volunteers were interviewed post experiment to gather knowledge about their satisfaction with the negotiation system and the systems usability. The questions asked were as follows:

1. With all successful negotiations, how satisfied were you with the agreed prices at the end of the negotiations? (1-Extremely dissatisfied; 4-Indifferent; 7-Extremely satisfied)
2. With all successful negotiations, what did you think of the final agreement, in relation to your goal for the negotiation? (1-Much worse than expected, 4-As expected; 7-Much better than expected)
3. Did you ever feel like you "lost" in any of the negotiations? (1-Not at all; 4-A moderate amount; 7-A great deal)
4. With all successful negotiations, do you feel the final agreements were fair? (-5 Agreements heavily biased towards seller, 0 Agreements were fair, +5 Agreements heavily biased towards yourself)

5. How important was the final price to your satisfaction with the system (1-Not at all; 4-A moderate amount; 7-A great deal)
6. Did you feel like the system kept you informed on what was going on, and within reasonable time? (1-Not at all; 4-A moderate amount; 7-A great deal)
7. Did the system support ways for you to undo any mistakes you made? (1-Not at all; 4-A moderate amount; 7-A great deal)
8. Do you feel the system did a good job of preventing you from making mistakes? (1-Not at all; 4-A moderate amount; 7-A great deal)
9. Was the system intuitive, did everything function the way you expected it to? (1-Not at all; 4-A moderate amount; 7-A great deal)
10. Do you think the design was minimalistic, only showing what was relevant to the situation? (1-Not at all; 4-A moderate amount; 7-A great deal)

4.6 Reflection on Development

Both PHP and Angular proved to be effective technology to develop the project with. Angular allowed the front-end development to be a lot quicker than initially expected. The ability to make each piece of the front-end a component, and so in turn to develop each piece in isolation of the others, made the development process much simpler. The fact that the negotiation systems front-end could be made after the ecommerce store had already been built and could be integrated so effortlessly into the already existing project is a testament to how powerful and helpful the Angular technology is. Back end development with PHP. PHP was very simple and straightforward to use. Its PDO functionality made reading and writing to the database very quick and easy. However, developing the negotiation system was quite tricky due to the stateless nature of PHP. As PHP is designed to receive requests and respond to them, keeping track of which request belonged to which negotiation was more convoluted than anticipated. This stateless nature required a design that would have to figure out and load up its current state every time a negotiation message was received from the front-end. This involved finding which negotiation the message belonged to and where in the sequence of negotiations messages the newly received message belonged. This process had to repeat itself every time an offer was sent to the system. This effects of this began to show when some parts of the experiment were executed. One part of an experiment required the system to take part in 200,000 negotiation. This took around 11 hours to execute. Debugging PHP proved to be a bit of an issue. With Angular, printing to the console could help solve most bugs. With PHP, outputting to the Apache logs seemed to be the only option, which was a time-consuming process. In the end a logging functionality was created to output debug statements to a file located in the PHP folder. This helped greatly. Finally, feature driven development proved to be a very suitable development process to use and follow. Different pieces of the project needed to be developed and tested in an iterative fashion, so feature driven development allowed for this perfectly.

4.7 Conclusions

In this chapter, the implementation of the front and back end of both the ecommerce site and the negotiation system was documented. An eCommerce site was implemented so the negotiation system could be given the correct context. A basic and an advanced negotiation system were implemented so the negotiation results that could be achieved with and without the use past negotiation and customer data could be seen clearly. Various screenshots highlighted the main code blocks for each piece of the project's main functionality. The use of PHP, Typescript, HTML and MySQL in these code blocks demonstrated the use of each within the context of the research project. A system to aid in the evaluation of the negotiation system was described before the two experiments, one which relied on the evaluation system, were described in greater detail, detailing exactly how they were implemented and undertaken. The main experiment involved the creation of mock customers to negotiate with the system to generate a dataset of negotiations that could then be evaluated. The second experiment used volunteers to gather information on the usability of the negotiation system. Finally, a reflection on the development process was given. Using Angular for the front end proved to be the correct decision as the ease in which the negotiation system could be integrated with the eCommerce site was a direct result of Angular's component-based architecture. Using PHP for the negotiation system, though effective as allowing for the implementation of the negotiation system, proved to be an inefficient choice due to the added overhead brought on by its stateless nature.

5 CHAPTER 5. EVALUATION OF THE NEGOTIATION SYSTEM

5.1 *Introduction*

In this chapter the results of the two experiments detailed in Chapter 4 will be described, analysed and evaluated. The first experiment is the main experiment, and it is a quantitative one, and the second experiment is an ancillary one, and it is qualitative. The first experiment is the main experiment and it aimed to evaluate the negotiation system itself through 4 different steps. These steps would help to provide insight into how the negotiation system performed with increasing customer counts, negotiation counts and pre-set store owner limits. After completion of the four steps, the results presented paint a clear picture on whether the system could be satisfactory to an eCommerce store owner. The second experiment is an ancillary experiment and is aimed at evaluating the customers' reaction and experience using such a system. Before the experiments were ran, the results of the system tests ran throughout development are show and evaluated as they proved to be very useful with providing detailed knowledge on the impact on the addition of each new feature to the system.

5.2 *Calibration - Results and Evaluation*

Throughout the development of the negotiation system, the negotiation evaluator was used for extensive testing after each new feature was implemented. These tests not only ensured each new feature was working correctly, but also revealed just how much of an impact the inclusion of each new feature had on the results the negotiation system was able to obtain.

Six different versions of the system were tested with the negotiation evaluator over the course of the development. This range of versions represent an evolution of the negotiator from Version 1, a "dumb" negotiator that simply sticks to the pre-specified limits and doesn't learn from the previous transactions, to Version 6, a very sophisticated negotiator that considers a wide range of environmental and contextual factors to achieve a best price negotiation.

Version	Features of the Version
1	System sticks to the store owner limit during its negotiations.
2	Version 1 + each successful negotiation with a customer makes the negotiation result the goal for the next negotiation with the customer.
3	Version 2 + addition of 1% of the product price on top the successful negotiation result for each successive successful negotiation with a customer
4	Same as version 3, but addition of 4% used instead of 1%.
5	System uses the customers max percentage of product price they are willing to spend during negotiations as its first and final counter offer in any future negotiations with customer, if the max is discovered.
6	Version 5 + if customer sends a final offer, accept it if it's above the store owner limit, regardless of the systems current goal based on the increment from version 3.

Table 5 Versions of the negotiation system

Using each of the above versions of the negotiation system, the negotiation evaluator system ran the same test 5 times: 10 customers of each of the 16 customer types negotiated 10 times each on a product worth €1000. The negotiation system had a store owner pre-set limit of €600 for each negotiation, regardless of version. Each customer type had a goal that was above the limit of the of the store owner, meaning the system should achieve a sale during every negotiation.

5.2.1 Results

As each version of the system was given the same limit while being tasked with the same amount of negotiations, with the same number of customers, all on the same product, the results each achieved can be directly compared to gauge the effectiveness of each version of the system.

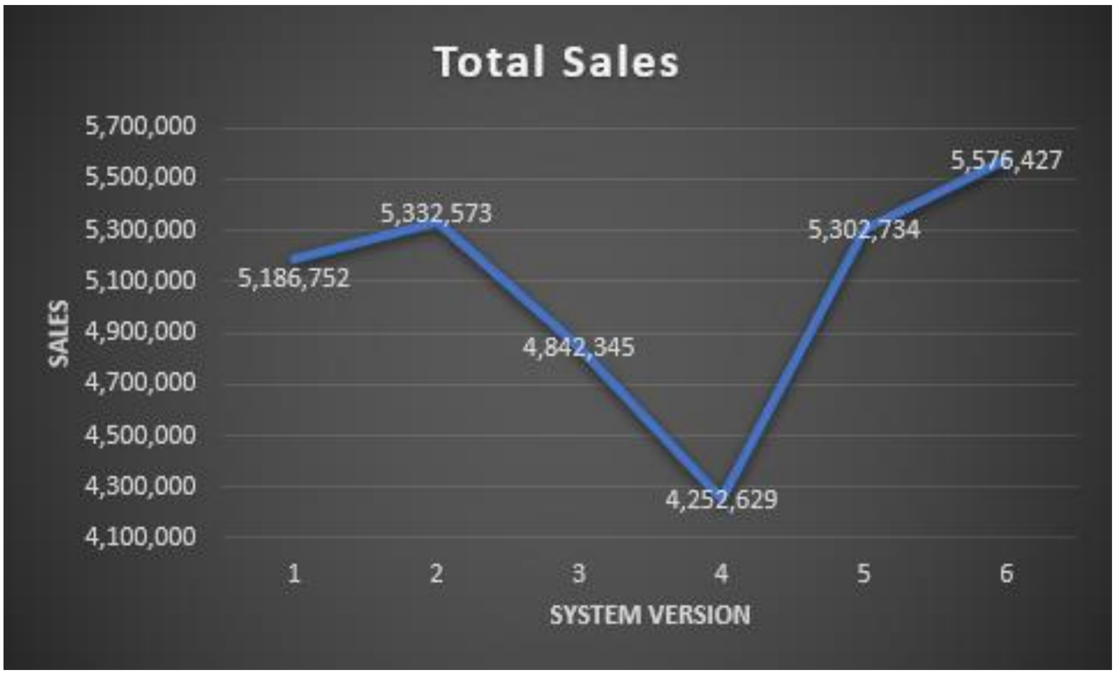


Figure 60 Monetary value of all combined sales for each system version

In Figure 60, the result of each negotiation (i.e. the final agreed amount between the customer and the negotiation system) the system engaged in for every negotiation with each customer of every type were added together to see how much revenue the negotiation system would have made for the store owner if the negotiations were with real customers for real products.

- Version 1 made ~ 5.2 million worth of product sales. This was the most basic version of the system and so could be treated as the baseline.
- Version 2 was made to be a simple improvement to Version 1, so it only managed a slight increase in of ~3%.
- Version 3 saw a decrease in performance from Version 1 (~6.5%) and an even more drastic decrease in performance from Version 2, achieving ~10% less in revenue for the store owner in comparison.
- This trend in decreased performance continued with Version 4. When compared with system Version 2 (which had achieved the best results at the time), Version 4 made ~20% less for the store owner.
- It wasn't until Version 5 that the performance began to rebound. Version 5 made almost 25% more than Version 4, but still underperformed when compared to Version 2 (albeit by a very small amount, <1% difference).

- Version 6 made 5% more for the store owner than Version 5 but more importantly, was the first version of the system that outperformed Version 2, achieving ~4.5% more in total revenue for the store owner in comparison.

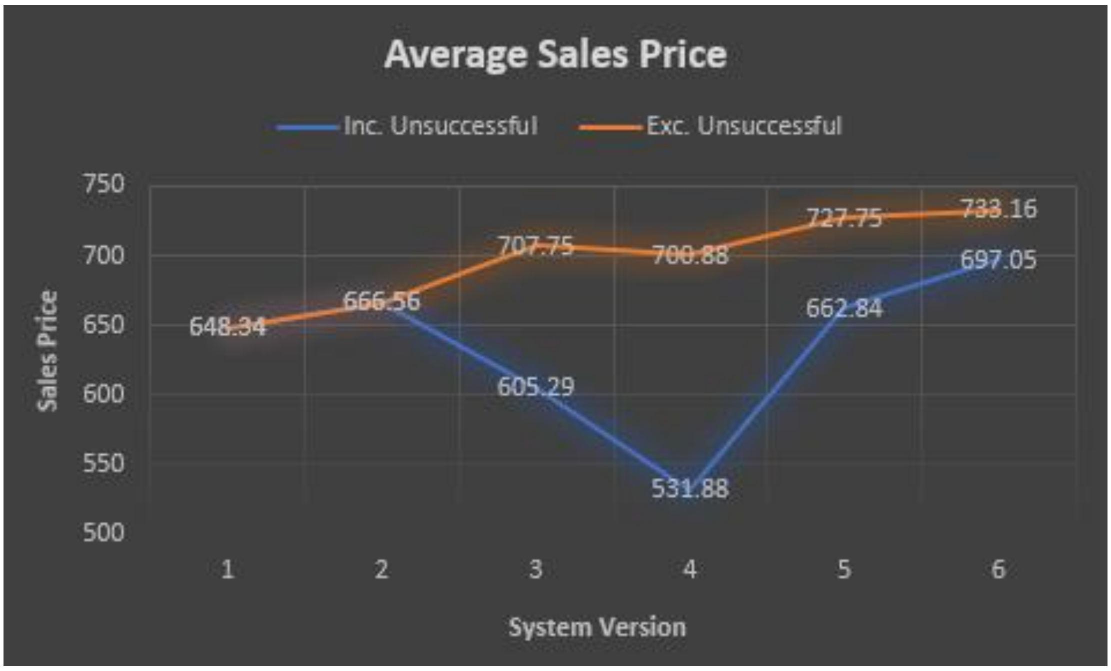


Figure 61 Average sales price per system version

Each negotiation on the product worth €1000 had to either end with a successful sale of the product for a price between €600 (the store owners limit pre-set in the system) and €1000 (the products asking price) or end with a rejection of a final offer from either the customer or the negotiation system. These rejections were counted as sales of €0, as by right it's a missed sale since both the customers and negotiation systems had goals that were compatible with each other's. Figure 61 shows the average price each system version attained for all the negotiations with all the customers of various types. The orange line represents the average sales price if the unsuccessful sales (rejections resulting in sales prices of €0) were ignored. The blue line takes these sales into account when calculating the average sales price for the system versions.

When the unsuccessful sales are not considered, the system version have a clear gradual increase in the average sales price they achieved. All versions, except from

Version 4 which had a 1% decrease in its average sales price than the previous system version, performed better than its previous version.

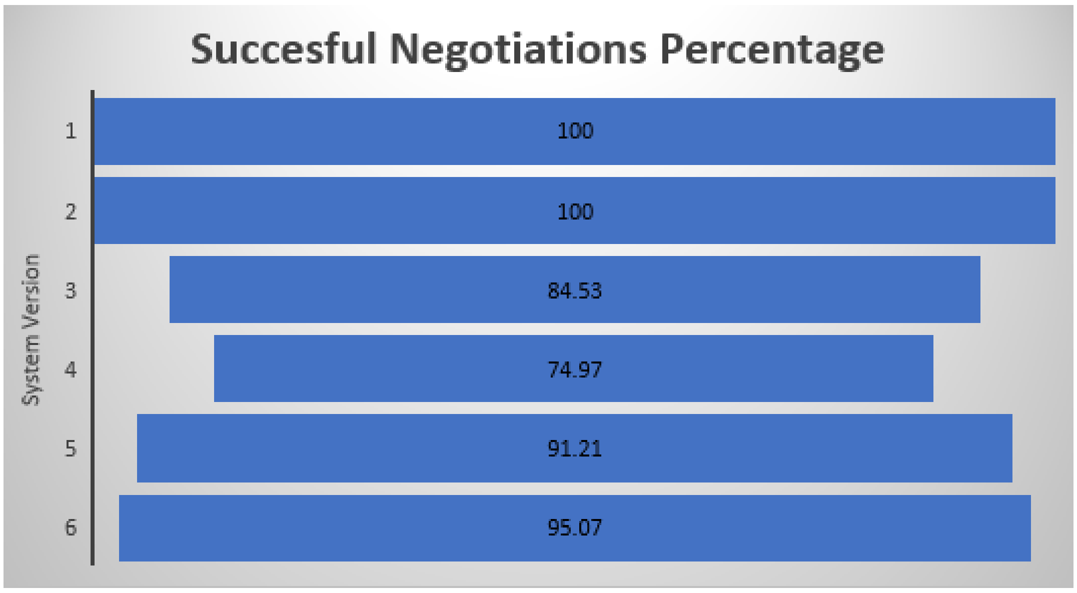


Figure 62 Percentage of negotiations that reached a successful ending per system version

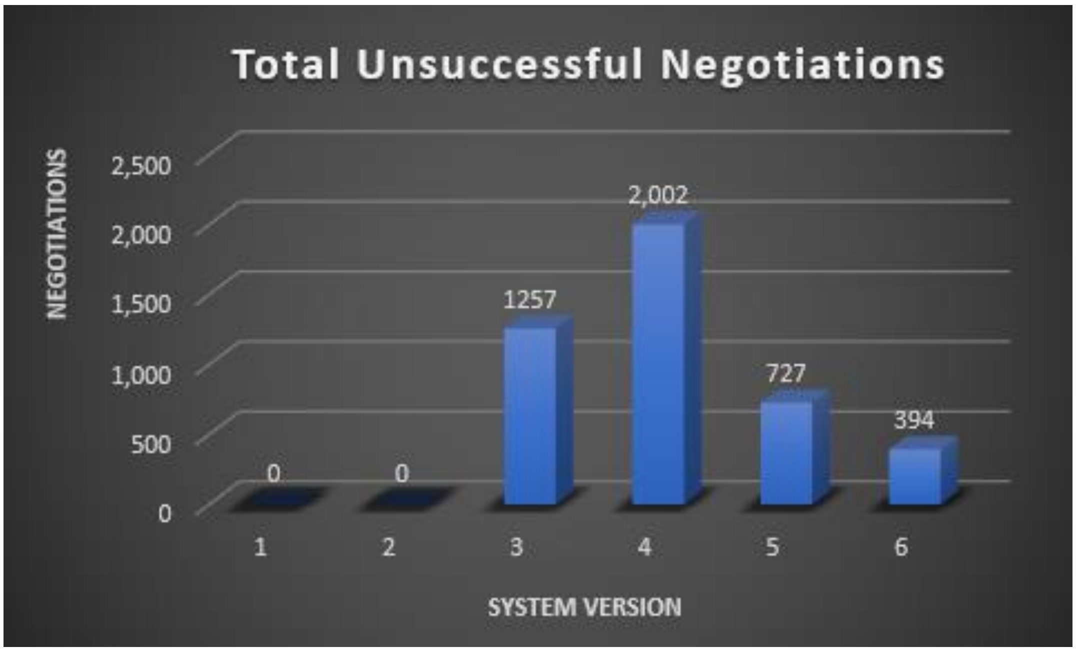


Figure 63 Number of negotiations that failed to reach a successful ending per system version

This is far from the same story when the unsuccessful negotiations are considered when calculating the averages. Versions 1 and 2 only had successful negotiations, but Version 3, despite its successful sales being higher on average, failed to reach a successful ending to its negotiations over 15% of the time (according to Figure 62), having over 1250 unsuccessful negotiations in total (according to figure 63). Version 4 performed even worse, after upping the increase to each successful negotiations' subsequent negotiation from 1% to 4%, over 25% of its negotiations were unsuccessful, whilst also having an average sales price ~20% lower (according to figure 61) than Version 2. Version 5 performed much better than Version 4 on all accounts. Its average sales price (not including unsuccessful negotiations) was almost 4% higher and when unsuccessful negotiations are considered that number jumps to around 20%. It had 1275 more successful negotiations (according to Figure 63), around a 16% increase in successful negotiations. The final version of the system, Version 6, did even better than Version 5. Version 6 achieved an average sale larger than Version 5 both when unsuccessful negotiations were and were not considered. It achieved 4% more successful negotiations than Version 5 and was the first version to achieve a higher average negotiation sales price than Version 2 when the unsuccessful negotiations are included.

The results presented throughout this section allow for an evaluation of the different Versions of the system.

5.2.2 Evaluation

For the longest time throughout development, it seemed version 2, despite using an extremely simplistic negotiation strategy, would achieve the best results for the store owner, as it attained the highest combined sales amount with the customers. Although Versions 3 and 4 achieved higher average sales among their successful negotiations by implementing the 'addition to each successful negotiation' goal-based behaviour, they did not exceed or even reach the same amount of revenue with their customers as Version 2, as they each had large amounts of unsuccessful negotiations, which were missed opportunities for sales. Although Version 2 had lower than average successful sales than Versions 3 and 4, the fact it always reached a deal with its customers more than made up for it. What became clear was that despite each new version of the system completing negotiations with higher prices with customers than their previous versions, they showed that the increase in not coming to agreements with customers as often has drastic impacts on the final combined sales amount that their system versions could achieve for the store owner.

Implementing the customer max functionality and using it in Version 5 resulted in a drastic reduction in the number of unsuccessful negotiations. This allowed Version 5 to achieve results that were very close to Version 2; averaging a sales price of €662.84 vs €666.56 and total combined sales of €5,302,734 vs €5,332,573, despite 8% of Version 5's negotiations being unsuccessful. Version 6 showed that better results can be achieved if at times an offer above the store owners limit is accepted despite it being lower than the systems goal. When the system didn't always prioritize trying to get as much from the customers as possible (Version 6), the results were better across the board, despite 5% of the negotiation being unsuccessful.

This does not mean Version 2 and its results should be discounted. Testing System 2 revealed that making the goal for each subsequent negotiation with a customer equal to the final sales price of the previous negotiation resulted in positive results for the store owner. For such a simple strategy, it resulted in 100% successful sales and 2.8% higher average sales than version 1 of the system which only had one goal; stick to the store owners' limit.

5.3 Store Owner Satisfaction Experiment

This experiment was broken down into 4 individual steps which aimed at investigating the impacts on the negotiation systems performance when

Step 1: the increment value was increased.

Step 2: the number of customers was increased.

Step 3: the number of negotiations was increased.

Step 4: the pre-set store owner limit was increased.

5.3.1 Impacts of Increasing the Increment Value

The difference in the results between Version 3 and 4 achieved above shows how important the choice of increment is to achieving satisfying results for a store owner. In the negotiation system, the increment is the percentage of the product price that is added on top of the sales price of any successful negotiation, so to provide a new, higher goal for the system to aim for the next time it negotiates with the customer who was involved in the successful negotiation. Therefore, before going forward with the other 3 steps in the experiment, an increment value to use during the next 3 steps must be decided on.

In this step of the experiment, Version 6 of the negotiation system was used with an ever-increasing increment value. The negotiation evaluator system was used 5 times for each of the different increment values, with the results being averaged and shown below. For each 10 customers of each of the 16 customer types negotiated 10 times on a product worth €1000. The negotiation system had a store owner pre-set limit of €600 for each negotiation. Each customer type had a goal that was above the limit of the of the store owner, meaning the system should achieve a sale during each negotiation.

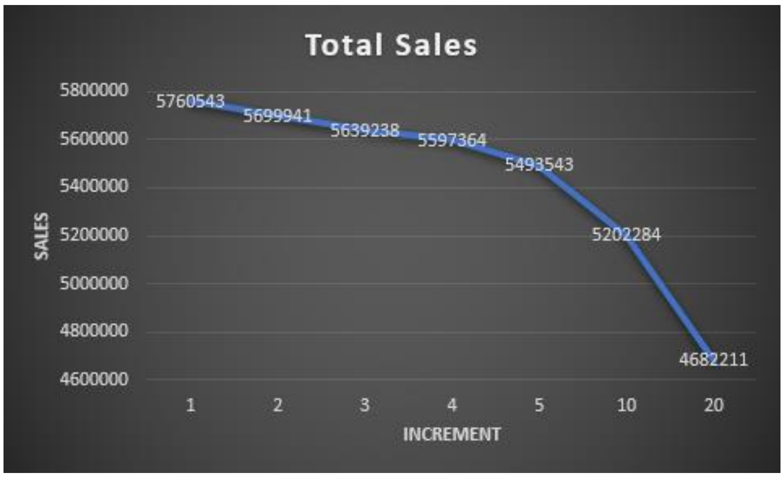


Figure 64 Total revenue from sales from successful negotiations with customers

When the total sales achieved when changing the increment percentage are compared, a steady decline is seen. Increasing the increment percentage between successful negotiations had a clear negative impact on the total sales the system can achieve when negotiating with its customers. Increasing the increment from 1% to 5% shows a decrease in total sales revenue by 4.7%. Increasing the increment by a further 5% shows a decrease in total sales by 5.3%. What's interesting is the correlation between the increase percentage and the total loss in sales compared to the 1% increment. Setting the increment to 10% results in sales revenue of around 10% (9.7%) less and setting the increment to 20% results in sales revenue of around 20% (18.8%) less.

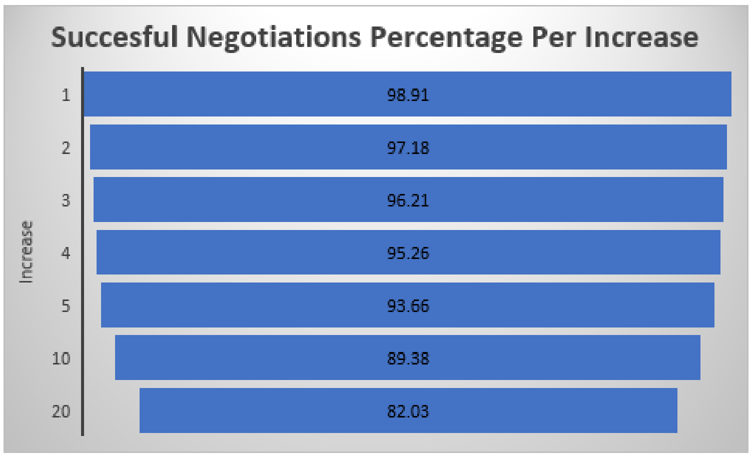


Figure 65 Number of successful negotiations after each increment increase

Figure 65 shows exactly why the decline in sales revenue occurs when the increments are increased. As increments are increased, the number of successful negotiations the system can achieve declines steadily. Interestingly, there seems to be a correlation between the percentage of successful negotiations that reduces during each increment increase and the number the increment is increased by. Each time the increment percentage increases a single percentage, there is on average around a single percentage decrease in the amount of successful negotiations the system can attain. Increasing the increment to 20% even results in around a 20% (~18%) decrease in successful negotiations.

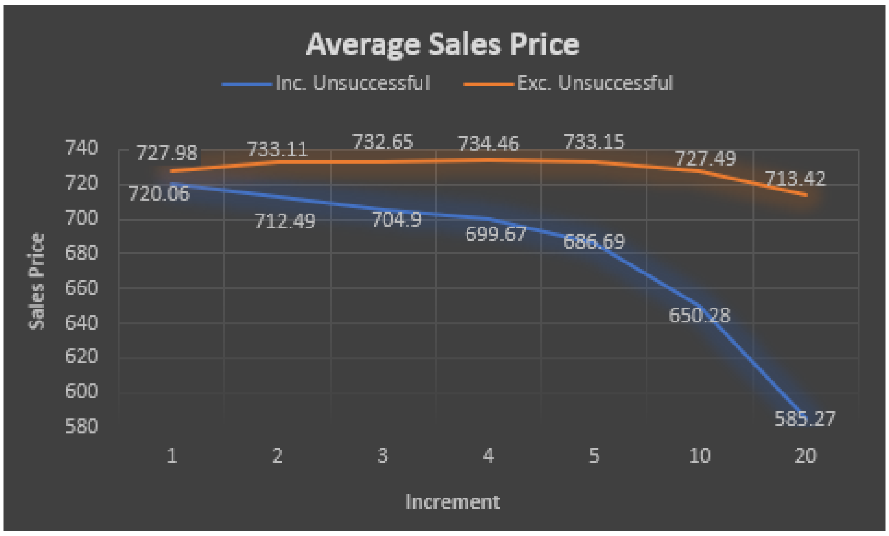


Figure 66 Average sales price after each increment increase

Figure 66 paints the same picture as the previous two Figures. The average sales price gets lower and lower the higher the increment percentage gets. Upping to 5% alone reduces the average sales price the system negotiates with the customer by around 5% (4.7% to be exact). This is expected given the decline in total revenue and percentage of successful negotiations. What is not expected is when only the successful sales are considered (orange line), there is a decline there as well from increment percentage of 4% onwards. It would be expected for the sales price to decrease (when unsuccessful sales are included) as there is a decline in the successful negotiation percentage, but the fact that even just the successful negotiations don’t see any benefit to the increment increase is very surprising. Not only is there not an increase in the sales prices of the successful negotiations only, but from 1% increment to 20% increment there is a decrease of 2.1% in the average sales price of the successful sales.

5.3.2 Impacts of Increasing the Amount of Customers

In this step of the experiment, the number of customers the system had to negotiate with were incremented to evaluate how the system handled different customer amounts. As the number of customers increased, the number of negotiations the system had to complete also went up. Therefore, the amount of sales was not be comparable between each run. Due to each customer only negotiating once each, there were no unsuccessful negotiations. Using Version 6 of the negotiation system with an increment amount of 1%, the negotiation evaluator system was used 5 times for each of the different customer numbers, with the results being averaged and shown below. For each customer, one of each of the 16 customer types negotiated on a product worth €1000. The negotiation system had a store owner pre-set limit of €600 for each negotiation. Each customer type had a goal that was above the limit of the store owner, meaning the system should have achieved a sale during each negotiation. Each customer only negotiated once so the increment to each subsequent successful negotiation was not a factor for this step in the experiment.

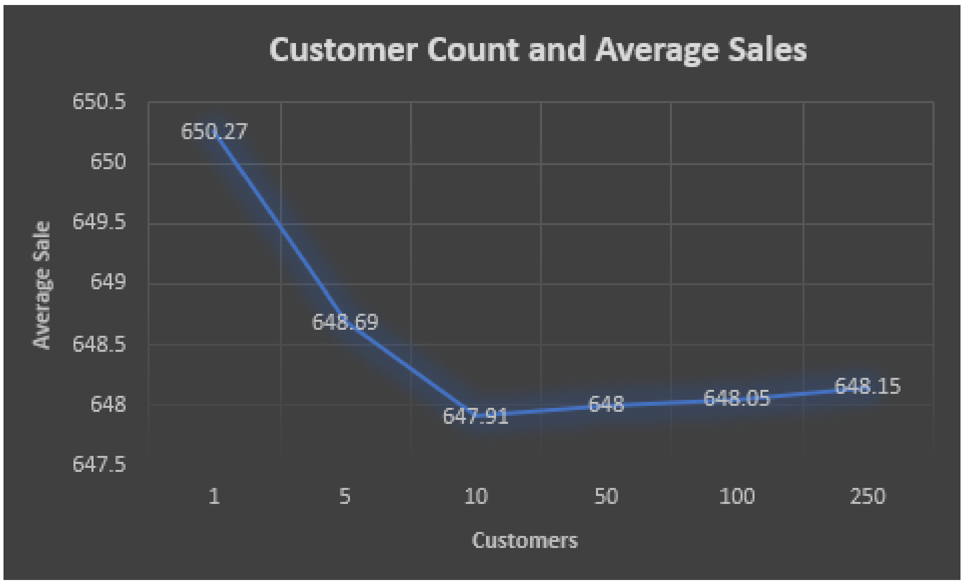


Figure 67 Average sales with increasing customer counts

Figure 67 shows that the negotiation system produces average sales per customer with larger variance when the number of customers who negotiate with the system is low. There is a 0.4% difference between the results achieved by 1 customer versus the results achieved by 10 customers, whilst the difference between 50 customer and 250 customers is 0.02%. The results also demonstrate that the system does not treat each customer differently to any degree of statistical significance. The difference between the average of one customer when compared with the average of 250 customers is only 0.33%. There is a very slight trend upwards after the system gets to and above 10 customers, the growth between the results with 10 customers and 250 customers is 0.04%. The difference is slight enough that it could be just circumstantial but the fact it goes in an uptrend and at no point in a downtrend is noteworthy. As each customer only ever negotiated once, the feature which causes the system to increase its desired amount after each successful negotiation had no impact on the results. The system learning the customer max and using it to get the highest sales price for each negotiation with not a factor in this step in the experiment due to only have one negotiation with each customer. What is interesting is that even without help from the two main features integrated to provide better results for the store owners than just the sticking to the limit, the system still averaged sales around 8% above the limit when negotiating with 250 customers of 16 types, 5 times.

5.3.3 Impacts of Increasing the Amount of Negotiations

In this step of the experiment, the number of negotiations the system had to have with each customer was incremented to evaluate how the system handled different negotiation amounts. Using version 6 of the negotiation system with an increment version of 1%, the negotiation evaluator system was used 5 times for each of the different negotiation numbers, with the results being averaged and shown below. One of each of the 16 customer types negotiated on a product worth €1000. The negotiation system had a store owner pre-set limit of €600 for each negotiation. Each customer type had a goal that was above the limit of the of the store owner, meaning the system should achieve a sale during each negotiation.

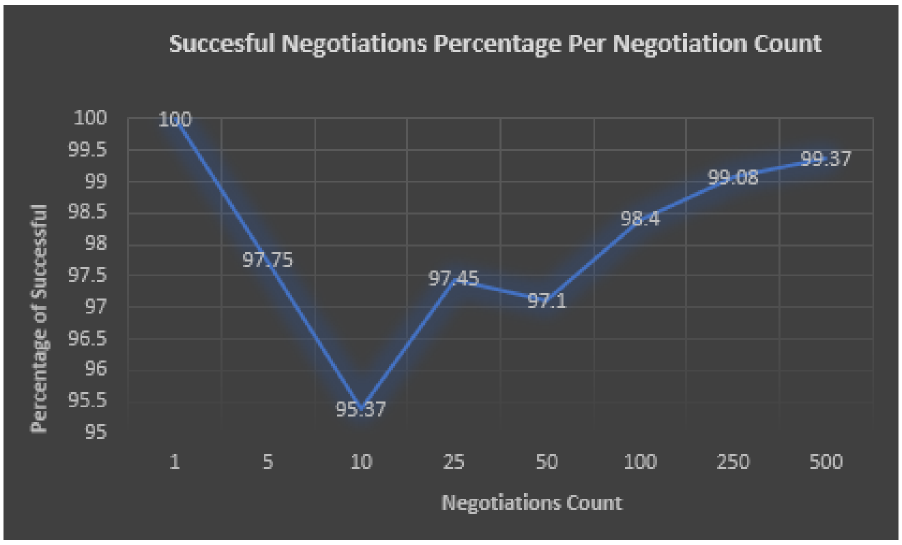


Figure 68 Percentage of negotiations that were successful after increased negotiation count

When each customer negotiated with the system a low number of times, the system demonstrates a decline in the amount of successful negotiations it reached. The was a little less than 5% decrease in the successful negotiations reached when customers negotiated with the system once vs 10 times. At this point, the declining trajectory of unsuccessful negotiations reversed. As each customer negotiated with the system from the 10th time onwards, the system produced less and less unsuccessful negotiations. By time each customer had negotiated with the system for the 250th time, over 99% of all negotiations were reaching successful conclusions.

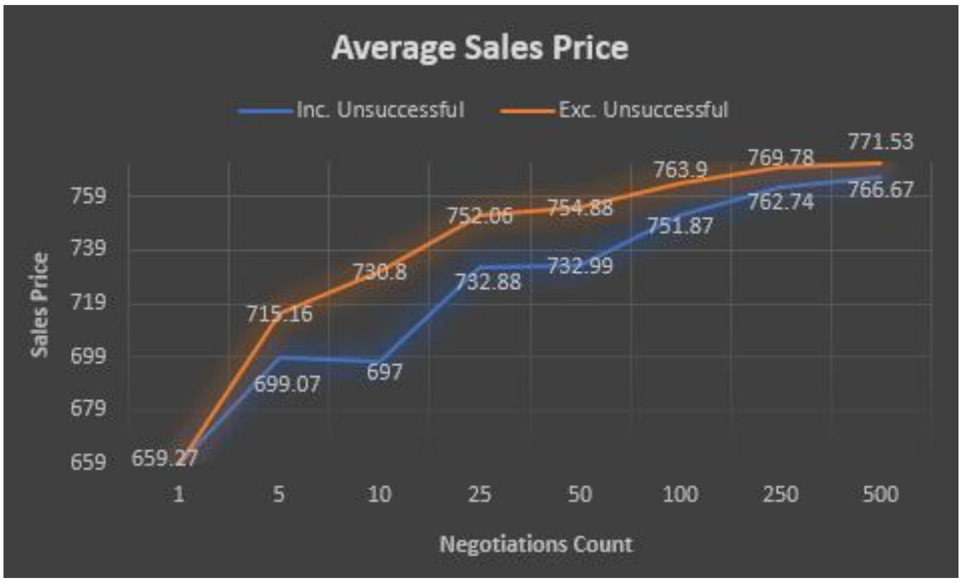


Figure 69 Average sales price per increased negotiation count

Figure 69 also shows positive results for the store owner the more that the customers negotiate with the system. Both all negotiations and only the successful negotiations showed a positive upwards trend as the number negotiations each customer had with the system increased. At only 10 negotiations, the negotiation system was averaging sales of €697. At 500 negotiations, that number had increase by ~10%. When unsuccessful sales were excluded from the average sales, a 5.5% increase showed between 10 and 500 negotiations. The results in unsuccessful negotiations accounting for only 1% of all negotiations when customers reach 500 negotiations shown in figure x explains why there isn't much of a difference between the average sales price when unsuccessful sales are considered versus when they are not. The difference between the two gets lower and lower as the negotiations count per customer increases. At on 10 negotiations, there is a difference of 4.6% while at 500 negotiations, the difference is only 0.7%.

5.3.4 Impacts of Increasing the Store Owner Limit

In this step of the experiment, the store owners limit that the system had to adhere to was gradually incremented to evaluate if the system could maintain an average sales price that wouldn't go below the limit, regardless of if the customers the system had to negotiate with had a goal that was lower than the limit that was set. For this step, only concerned the average of the successful sales are of concern. Some of the customers will have goals that are below what some of the limits are and so unsuccessful negotiations will occur. We will not include these in the average sales as we will learn nothing from them. This step uses version 6 of the negotiation system, the negotiation evaluator system was used 5 times for each of the different customer numbers, with the results being averaged and shown below. 5 customers of each of the 16 customer types negotiated on a product worth €1000, 5 times each.

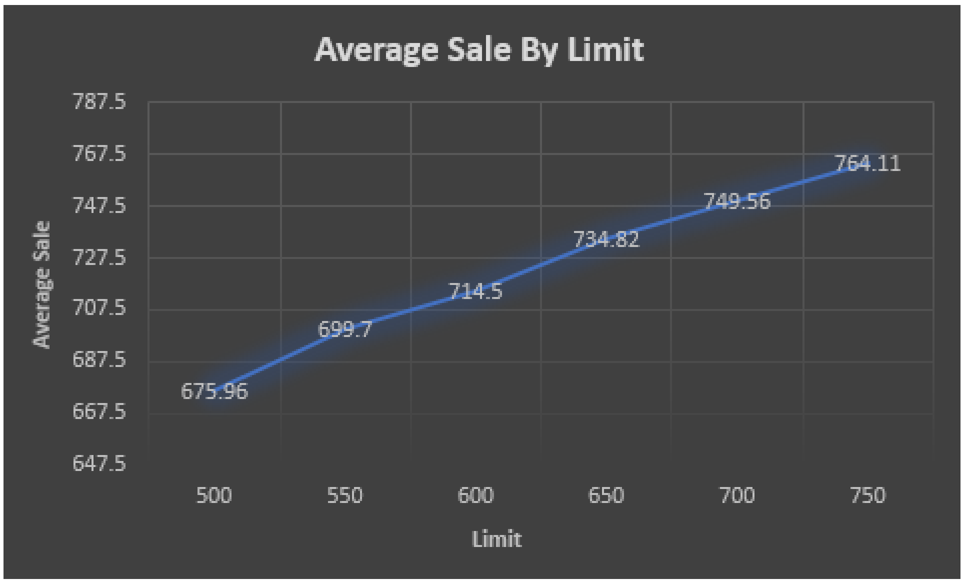


Figure 70 Average sale as store owner limit was increased

The figure above shows that at no point did the system ever fail to stick to its set limit, regardless of how low or high it was set. Regardless of what the limit was set to, the system always not only adhered to it, but outperformed it. The higher the limit was set, the higher the average sales price reached. However, as the limit got closer and closer to the product price, the percentage above the limit the average sales price got was smaller and smaller each time. With a limit of €500, the average sales price was a little over 35% above the limit, but with a limit of €750, the average sales price was only 8.5% above the limit.

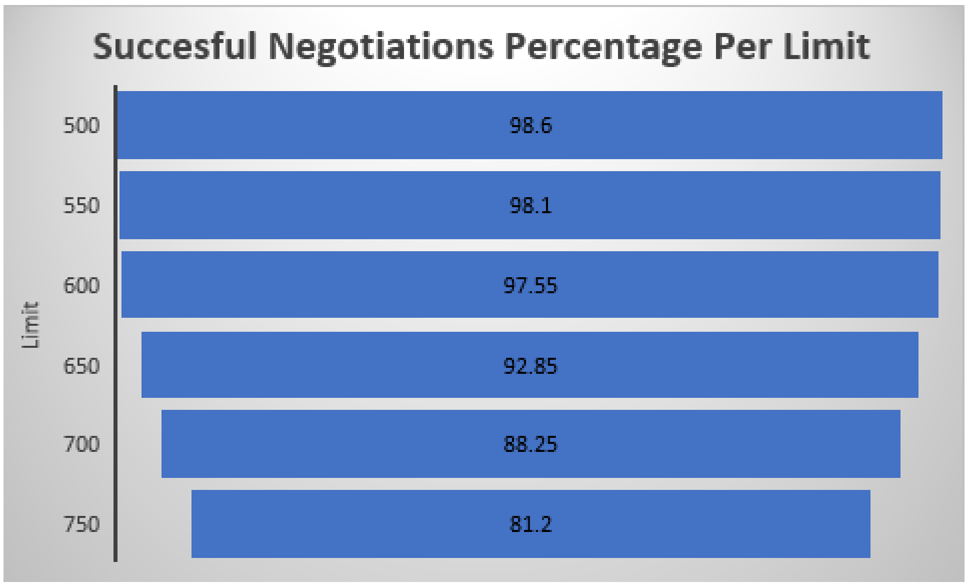


Figure 71 Percentage of successful negotiations as store owner limit was increased

Figure 71 demonstrates that as the store owner limit gets increased further and further towards the product price, the number of negotiations the system can reach a successful conclusion for get reduced. When the limit was set to €500 (50% of the product price) the system only failed to reach a successful conclusion in 1.4% of its negotiations. When the limit was 50% and was increased to 55%, there was only a 0.5% increase in the amount of negotiations failing to reach a successful conclusion. However, when the limit was set to €700 (70% of product price) and there was the same increase of 5% to bring it to a limit of 75% of the product price, there was a 7.05% increase in the amount of negotiations failing to reach a successful conclusion. This shows that the limits the store owner set for the system have more and more of an impact on the percentage of successful negotiations the higher up they go. Going from a 50 – 55 percent limit produced far less unsuccessful negotiations than going from 70 – 75%.

5.3.5 Key Findings

Now that the above experiment has been completed, the results allow for several findings have been uncovered.

1. The negotiation system not only can handle an increasing number of negotiations, but the results it achieves get better as the number of negotiations with each customer increases. Both the number of successful negotiations and the average agreed amount between the customers and the system increase as more negotiation occur. This could be considered a double-edged sword, however. As this improvement in results as negotiations increase means for the system to attain the best results, the customers must negotiate with it many times. The results attained when the negotiations count is relatively low (<10 negotiations) do provide results that would be deemed satisfactory to the store owner though, as they are above the limit, by 8% on average when negotiating with the 16 types of customers generated for the experiment.
2. The negotiation system proved be able to handle large amounts of customers. The system provided results when having to negotiate with one versus 250 customers with a difference of less than 0.3%, with evidence of a very slight uptrend after negotiating with 10 customers. When gathering these results, the customer max and increment to each successful negotiation were not used as each customer only negotiated once. Despite this, the system produced negotiation results that were again, 8% above the store owners limit on average.
3. The experimental results indicate that adding an incremental percentage to the result of the previous negotiation (to create a price goal for the next negotiation) does not generate satisfying results for the store owner. Using an increment proved to lower all of the following: sales revenue, the number of successful negotiations and the average sales price of the negotiations. The only benefit that could be taken from using the increments is the fact that by upping the negotiation goal after each successful negotiation with a customer, the system can find out the customers max below asking quicker. The tests ran during the development of the system showed how much of a benefit learning

the customer max is, as Version 5 (the version which began using customer max) had results that were 25% better than before using the max.

4. The negotiation system is fully capable of sticking to and respecting the store owners limit. Once a limit is set by a store owner, the system will never agree to sell a product for a price that breaks the limit. The experiment did reveal that as the percentage of the product that the limit gets set to increases, the system fails to reach as many successful conclusions to negotiations. The experiment also revealed that the lower the limit is set, the higher above the limit the average of each sale gets. With a product worth €1000, the system on average negotiated on a sales price over 35% above the limit if the limit was set to 50%. When the limit was 75%, the average sales price was only 8.5% above the limit.

5.4 Customer Satisfaction Experiment

The set of 10 questions outlined in Section 4.5.2 were used to evaluate both the usefulness and the usability of the system in a qualitative manner.

5.4.1 Participants

Five participants were invited to have multiple engagements with the negotiation system using the eCommerce site. After which, the participants provided feedback via semi-structured interviews. The interview questions focused on getting feedback on the usability of the negotiation system integrated with the eCommerce site and their satisfaction with their experiences negotiation with the system.

Code	Age	Gender	Familiarity with eCommerce
F1	Early 20s	F	Purchases from eCommerce sites most months
F2	Early 50s	F	No experience with eCommerce
M1	Mid 40s	M	Browsed eCommerce sites before but never purchased
F3	Late 20s	F	Purchases from eCommerce sites a few times a year
M2	Late 50s	M	Purchases from eCommerce sites once every few years

Table 6 Experiment participants

5.4.2 Experiment Results and Evaluation

The participants answer to questions 1 and 2 revealed that only some were satisfied with the results of the negotiations they had with the system. There was a noticeable trend among the level of satisfaction each participant had with their attained results and their age. F1 and F3 (the youngest participants) were the most satisfied with their results. F2 and M2 (the oldest participants) were the least satisfied with their results, despite having differing experience levels with eCommerce.

The answers participants gave to questions 3 and 4 provided insight into how revealing is the negotiation systems goal bias towards the store owner is. None of the participants felt like the system was set up against them, none felt like they "lost" in any of the negotiations, despite only 40% of the participants indicating in the prior questions that they were satisfied with the results they had achieved during the negotiations. All participants indicated during question 4 that they felt the system was fair.

The answers participants gave to question 5 revealed how important final price is to their level of satisfaction with the system. This would suggest that a negotiation system biased towards providing the best results to the store owner might not work out. If the customers final price is not seen as important to the system, this could be interpreted as a disregard for the importance of the customers satisfaction. However, responses to questions 3 and 4 showed that the participants found the system to be fair. This would suggest that no participants were able to detect that the system was designed to get the best result for the store owner only. If they had, the responses to question 5 would suggest that the answers to questions 1 and 2 would be far lower than they were. This brings up an interesting situation. Customers could potentially be very against a system that favours the results of the store owners, but if they are unaware of this, is this still an issue?

The second set of question were related to the usability of the system based on Nielsen's heuristics. Participant F2 and M2 had limited experiences with eCommerce. There was a trend with these participants to answer with scores 3 and 4 to the usability questions. This could be a default response; as they might not have known just how usable or unusable the website/negotiation system were. Age could have been a factor here, but since M1 is close to their age but has more experience, his scores are slightly higher for the usability questions, suggesting experience and familiarity with eCommerce played a factor when judging the systems usability, not age. This is supported further with F1 and F3 who had the most experience with eCommerce systems and scored the website and negotiation system high based on the usability questions scale. A lack of instructions was given to participants prior to using the system. Despite this, the participants themselves considered the design minimalistic. This is evidence of the systems usability.

5.5 Conclusions

Two separate experiments were undertaken to judge how much satisfaction the negotiation system could provide to both the store owner and its customers. Experiment 1, the main experiment, revealed that the negotiation system does meet the store owner’s satisfaction. Not only does the system stick to the limits provided to it, the system achieves higher sales on average than just the limit. The system was proved to be capable of maintaining its performance level when the number customers was increased and showed even showed an increase in performance when the amount of negotiations increased. Experiment 2, the ancillary experiment, revealed that the negotiation system provided satisfying results to participants, but only participants who have had prior experience with eCommerce. The experiment also revealed that the participants were unable to detect the system was biased towards the store owner, even despite the it not always providing the most satisfying results for them. In terms of usability, the system was deemed minimalistic and intuitive by the participants.

6 CHAPTER 6. CONCLUSIONS AND FUTURE WORK

6.1 Introduction

The aim of this research was to examine the possibility of using an automated negotiation system to negotiate with customers to enable flexible product pricing in eCommerce. The negotiation system would use past negotiation and customer data to help achieve satisfactory results for the store owner when it came to the prices the system would agree to sell the products to the customers for.

In this chapter, the conclusions gathered from each of the thesis chapters are summarised and the key findings from each chapter will be presented. Following this, the contributions of the findings to the existing body of work are then discussed so that their impact can be gauged. The findings will show a clear potential for a negotiation system such as the one presented in this research to be used in the context of eCommerce. The research will show that the results such a system can produce are enough to satisfy the store owner the system represents.

Finally, potential future work that can form a continuation to the research findings is suggested to allow for further research into the areas discussed and examined throughout the research project. The potential future work includes enhancements to the negotiation strategy used by the researched negotiation system and covers work that could be carried out to improve the evaluation process of such a system.

6.2 Conclusions

6.2.1 Negotiation Research

Chapter 2 revealed that an automated approach with software agents is the most common method of developing negotiation systems and so would be an appropriate route to take for developing a negotiation system for use in an eCommerce site.

Examining the existing research revealed the possibility of using past negotiation results and customer data as the backbone for the development of the negotiation system that mirrors real life negotiations more closely. The use of satisfaction as a method of evaluation for such a system from the perspective of a store owner was detailed and justified.

The key findings for chapter 2 were:

- The use of software agents for automation was found to be of extreme popularity within existing negotiation systems.
- It was very common among existing negotiation systems to use agents to represent both the buyer and seller.
- There was a gap in the research for negotiation systems that only represented the sellers.
- Further research was needed into negotiation systems which focused on individual outcome and not joint.
- The possibility of using past negotiation and customer data to aid in future negotiations is suggested in existing research, but not thoroughly researched.

6.2.2 System Design

Chapter 3 detailed designs for both the negotiation system and an eCommerce site for which to provide the system the correct context. Features that use data from the past negotiation results and customer patterns were included in these designs. Finally, experiments were introduced that would provide results from which the proposed system could be evaluated to determine if the proposed system would be satisfactory for eCommerce store owners.

The key findings for chapter 3 were:

- Feature driven development allows for each new feature to be tested after implementation, rather than at the end of development. This could lead to some interesting findings as the negotiation system is brought from using a simplistic strategy to the strategy that makes use of the past negotiation and customer data.

- Angulars' component-based architecture could be a perfect solution for the integration of the negotiation system into the context of eCommerce.
- PHP would be an appropriate choice of technology for the negotiation system due to its support for MySQL through PDOs.

6.2.3 System Implementation

Chapter 4 provided in depth descriptions for the implementation of the designs detailed in chapter 3. Through code blocks and explanations, the inner workings of both the eCommerce site and the proposed negotiation system were provided. Close attention was given to how the system used past negotiation and customer pattern data. How both experiments suggested in the previous chapter were undertaken were described before finally giving a detailed review of the development process.

The key findings were:

- Both Angular and PHP were capable tools for implementing the eCommerce site and the negotiation system.
- Angular proved to be a huge aid in the integration of the negotiation system into the eCommerce site due to its component-based architecture.
- Using PHP for the negotiation system resulted in an inefficient system due to its stateless nature.

6.2.4 System Evaluation

Chapter 5 evaluated the negotiation system implemented in chapter 4 with the two separate satisfaction level gathering experiments. Multiple versions of the negotiation system were compared so to see the impact each new feature implementation had on the results the system could achieve. The results of the experiments proved that the system provided satisfactory results to customers who had previous experience with eCommerce. The negotiation system was able to provide results that were to the satisfaction of the store owner as the negotiation system never failed to maintain the product limit given to it.

The key findings were:

- The negotiation system was successfully able to be integrated into the context on an eCommerce site.
- Using the evaluator system, the system version which used the basic negotiation strategy was proven to be less effective at achieving satisfying results than the version which used the advanced strategy.
- Using past negotiation and customer pattern data, the system was able to achieve sales that were 8% above the limit on average.

6.3 Contributions and Impact

The contributions to the existing body of work that this research project consists of include:

- The research examined the performance of a system that focuses on individual outcome in negotiation, instead of the more heavily researched joint outcome. Most of the existing body of research focuses on negotiation systems that attempt to aid both the buyers and sellers to reach mutually beneficial (‘joint’) outcomes. The research here focused specifically on creating a negotiation system that aided the seller exclusively to reach an outcome beneficial to them individually.
- The research added to the body of knowledge of using software agents in the context of negotiation systems. In this research project, only one software agent was used in the negotiation system, for the seller, instead of the much more common approach of using one or many agents to represent or aid both the buyers and the seller. This approach allowed for the research into focus the much less explored human to agent negotiation systems, rather than agent to agent negotiation systems.
- The research examined the potential for a negotiation system to exist in the context of an eCommerce site. Plenty of research exists for systems that enable

negotiation in the context of the internet. Research into systems that enable negotiation in the more specific context of eCommerce sites is much more limited. This research project focused on the development of a system that was directly integrated into an eCommerce site that was then evaluated.

- The research helped push forward the jump into using past negotiation data in negotiations. The idea of using past negotiation data, such as the results or rejections, to aid in future negotiations has been suggested as a topic for future work in multiple existing research projects. This research investigated the possibility of using the results of previous negotiations with individual customers as foundations from which to start future negotiations with those individual customers.
- The research investigated the potential of mirroring the negotiations that would take place in eCommerce more closely with the negotiations that would take place in stores in real life. As store owners can remember customers and their negotiation patterns, the negotiation system developed during this research attempted to do the same by learning each customers value for the max below asking price percentage they tended to aim for during their negotiations. This research project then evaluated the use of this value when being used by the negotiation system to try get the best deal possible during future negotiations with the customer.

6.4 Future Work

Several avenues for future research presented themselves at the cessation of this research project:

1. The negotiation system currently only learns each customer max below asking price value once for each customer. This works fine when customers have only one value for this which they never change. If a customer wished to be more dynamic and change this value either over time or on a product by product basis, the system has no way of dealing with this as it does not look to relearn the value once it is already found. Future work could deal with this limitation

by extending the current system to allow for the recalculation of the value for each customer on a dynamic basis.

2. Further work on using rejections in future negotiations could be looked at. The negotiation system researched for this project used rejections only as an indicator to reduce the systems goal for the negotiation with each individual customer. This could be expanded on further by making it so that if a customer rejects a final offer sent by the negotiation system, this offer is never again offered to the customer. A ‘ceiling’ value for each individual customer could be maintained by the system so that future offers don’t go above the ceiling i.e. make an offer the customer has a history with rejecting, thus throwing away a potential sale.
3. Further work could also be carried on creating a ‘floor’ value and having it be maintained by the system. The negotiation system as it currently stands restarts the process of creating a negotiation goal for a customer when the customer rejects an offer made by the system. After the goal for the customer has been continuously increased after each successful negotiation with the customer, it takes only one reject for the negotiation system to go back to the goal before any of the increases. If the system has a floor value, i.e. a value that represents an offer that the system knows the customer will accept, the system could retreat to this value as the goal after a rejection instead of using the store owners limit as the goal after a rejection. This has the potential to ensure all future successful negotiations after failed negotiations achieve a higher sales price than what they achieve now.
4. To combat the limitation brought on by needing to create mock customers to negotiate with the system to gather enough negotiation data to allow for accurate evaluation of the system, future work could include a step where a large group of volunteers are gathered to act as real customers. These customers could negotiate with the system enough times to build both a larger and more accurate dataset of negotiations. This would successfully combat the limitation in the evaluation of the researched negotiation system where only 16 types of ‘customers’ were created. 16 types of customers are nowhere near what is needed to get an accurate representation of the vast potential variance in negotiation strategy between potential customers.

5. As an alternative to the incrementally increasing goal that the research project implemented, future work could include creating a value for a goal based on feedback from real life store owners. The negotiation system researched for this project uses a dynamic limit that can be changed as the base goal of the system. Real store owners could be interviewed so to find out if there is a goal that is most common among them (e.g. 60% of the asking price is revealed to be a satisfactory goal for 85% of store owners). This goal could be used as the base goal in the system, in place of the limit.

BIBLIOGRAPHY

Amgoud, L., & Kaci, S. (2005, July). Strategical considerations for negotiating agents. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems* (pp. 1215-1216). ACM.

Angular Architecture. (2019). Retrieved from https://www.ngdevelop.tech/angular/architecture/

Angularjs architecture, core concepts - AngularJS example. (2019). Retrieved from https://codesjava.com/angularjs-architecture-overview-core-concepts-advantages-disadvantages

Ananthanarayanan, R., & Kumar, M. (2005, July). Negotiation support in online markets, with competition and co-operation. In *E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference on* (pp. 42-49). IEEE.

Awad, M. A. (2005). A comparison between agile and traditional software development methodologies. *University of Western Australia*.

Badica, C., Ganzha, M., & Paprzycki, M. (2006, June). Rule-Based automated price negotiation: overview and experiment. In *International Conference on Artificial Intelligence and Soft Computing* (pp. 1050-1059). Springer, Berlin, Heidelberg.

Badica, C., Ganzha, M., Gawinecki, M., Kobzdej, P., & Paprzycki, M. (2006). Utilizing Dutch Auction in an Agentbased Model E-commerce System. In *Proceedings of the 14th International Enformatika Conference, World Enformatika Society* (pp. 7-12).

Benoit, M., Anthony, R., & Wee, L. B. (1999). Feature-driven development.

Cao, M., Luo, X., Luo, X. R., & Dai, X. (2015). Automated negotiation for e-commerce decision making: a goal deliberated agent architecture for multi-strategy selection. *Decision Support Systems*, *73*, 1-14.

Chen, E., Vahidov, R., & Kersten, G. E. (2005). Agent-supported negotiations in the e-marketplace. *International Journal of Electronic Business*, *3*(1), 28-49.

Cheng, C. B., Chan, C. C. H., & Lin, K. C. (2006). Intelligent agents for e-marketplace: Negotiation with issue trade-offs by fuzzy inference systems. *Decision Support Systems*, *42*(2), 626-638.

Clarke, P., & O'Connor, R. V. (2012). The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, *54*(5), 433-447.

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer*, *34*(11), 131-133.

Cochran, D. (2012). *Twitter bootstrap web development how-to*. Packt Publishing Ltd.

Cohen, D., Lindvall, M., & Costa, P. (2003). Agile software development. *DACS SOAR Report*, *11*, 2003.

Curhan, J. R., Elfenbein, H. A., & Xu, H. (2006). What do people value when they negotiate? Mapping the domain of subjective value in negotiation. Journal of personality and social psychology, 91(3), 493.

Fathey, A., & Moawad, R. (2005, December). E-commerce agents and online negotiation process. In *Information and Communications Technology, 2005. Enabling Technologies for the New Knowledge Society: ITI 3rd International Conference on* (pp. 435-444). IEEE.

Geer, D. (2006). Will software developers ride ruby on rails to success?. *Computer*, *39*(2), 18-20.

Gosselin, D., Kokoska, D., & Easterbrooks, R. (2010). *PHP Programming with MySQL: The Web Technologies Series*. Cengage Learning.

Grandon, E. E., & Pearson, J. M. (2004). Electronic commerce adoption: an empirical study of small and medium US businesses. *Information & management*, *42*(1), 197-216.

Greer, D., & Hamon, Y. (2011). Agile software development. *Software: Practice and Experience*, *41*(9), 943-944.

Halpert, J. A., Stuhlmacher, A. F., Crenshaw, J. L., Litcher, C. D., & Bortel, R. (2010). Paths to negotiation success. Negotiation and Conflict Management Research, 3(2), 91-116.

Hansson, D. H. (2009). Ruby on rails. *Website. Projektseite: http://www. rubyonrails. org*.

Hibbs, C. (2005). Rolling with ruby on rails. *The Open Source Web Platform onlamp. com*.

Huang, P., & Sycara, K. (2002, January). A computational model for online agent negotiation. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on* (pp. 438-444). IEEE.

Jadhav, M. A., Sawant, B. R., & Deshmukh, A. (2015). Single page application using angularjs. *International Journal of Computer Science and Information Technologies*, *6*(3), 2876-2879.

Jain, N., Bhansali, A., & Mehta, D. (2015). AngularJS: A modern MVC framework in JavaScript. *Journal of Global Research in Computer Science*, *5*(12), 17-23.

Junyan, Z., Jiang, T., & Gang, D. (2007, November). Agent-based multi-factors adaptive negotiation in e-commerce. In *Grey Systems and Intelligent Services, 2007. GSIS 2007. IEEE International Conference on* (pp. 1528-1532). IEEE.

Karp, A. H., Wu, R., Chen, K. Y., & Zhang, A. (2004, May). A game tree strategy for automated negotiation. In *Proceedings of the 5th ACM conference on Electronic commerce* (pp. 228-229). ACM.

Kaur, R., & Sengupta, J. (2013). Software process models and analysis on failure of software development projects. *arXiv preprint arXiv:1306.1068*.

Kersten, G. E., & Lo, G. (2003). Aspire: An integrated negotiation support system and software agents for e-business negotiation. *International Journal of Internet and Enterprise Management, 1*(3), 293-315.

Kexing, L. (2011). A Survey of Agent Based Automated Negotiation. 2011 International Conference On Network Computing And Information Security, 24- 27. http://dx.doi.org/10.1109/ncis.2011.103

Kozlowski, P. (2013). *Mastering Web Application Development with AngularJS*. Packt Publishing Ltd.

KPI. (2019). Traditional vs. Agile Software Development Methodologies. Retrieved from http://www.kpipartners.com/blog/traditional-vs-agile-software-development-methodologies

Kulkami, S., Roy, B., & Iyer, S. (2017, April). E-negotiator based on buyer's surfing pattern. In *Communication Systems, Computing and IT Applications (CSCITA), 2017 2nd International Conference on* (pp. 48-53). IEEE.

Lalit. (2019). MySQL Architecture and Components. Retrieved from https://lalitvc.wordpress.com/2016/11/03/mysql-architecture-and-components/

Lau, R. Y. (2007). Towards a web services and intelligent agents-based negotiation system for B2B eCommerce. *Electronic Commerce Research and Applications, 6*(3), 260-273.

Lei, K., Ma, Y., & Tan, Z. (2014, December). Performance comparison and evaluation of web development technologies in php, python, and node. js. In *2014 IEEE 17th International Conference on Computational Science and Engineering (CSE)*(pp. 661-668). IEEE.

Luo, X., Jennings, N. R., & Shadbolt, N. (2003, September). Knowledge-based acquisition of tradeoff preferences for negotiating agents. In *Proceedings of the 5th international conference on Electronic commerce* (pp. 138-149). ACM.

Madsen, M., Tip, F., & Lhoták, O. (2015, October). Static analysis of event-driven Node. js JavaScript applications. In *ACM SIGPLAN Notices* (Vol. 50, No. 10, pp. 505-519). ACM.

More, A., Vij, S., & Mukhopadhyay, D. (2014). Agent Based Negotiation Using Cloud–An Approach in E-Commerce. In *ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India-Vol I* (pp. 489-496). Springer, Cham.

Najjar, A., Gravier, C., Serpaggi, X., & Boissier, O. (2016, October). Modeling User Expectations & Satisfaction for SaaS Applications Using Multi-agent Negotiation. In Web Intelligence (WI), 2016 IEEE/WIC/ACM International Conference on (pp. 399-406). IEEE.

Ngai, E. W., & Wat, F. K. T. (2002). A literature review and classification of electronic commerce research. *Information & Management*, *39*(5), 415-429.

Ngai, L., Mak, P., Ni, W. C., Liu, L. C., & Wu, C. (2007, October). A Semi-automated negotiation process to improve the usability for online marketplaces. In *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on* (pp. 253-258). IEEE.

Nielsen, J. (1995). 10 usability heuristics for user interface design. *Nielsen Norman Group*, *1*(1).

Ow, T. T., O'Neill, B. S., & Naquin, C. E. (2014). Computer-aided tools in negotiation: negotiable issues, counterfactual thinking, and satisfaction. *Journal of Organizational Computing and Electronic Commerce*, *24*(4), 297-311.

Pang, J., & Blair, L. (2004). Refining Feature Driven Development-A methodology for early aspects. *Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*, *86*.

Pease, W. (2001). E-commerce enabling technologies. *e-Commerce in Regional Australia Update 2001*.

Prokofyeva, N., & Boltunova, V. (2017). Analysis and practical application of php frameworks in development of web information systems. *Procedia Computer Science*, *104*, 51-56.

Pruitt, D. G. (2013). *Negotiation behavior*. Academic Press.

Raiffa, H. (1982). *The art and science of negotiation*. Harvard University Press.

Rahayu, R., & Day, J. (2015). Determinant factors of e-commerce adoption by SMEs in developing country: evidence from Indonesia. *Procedia-Social and Behavioral Sciences*, *195*, 142-150.

Rahwan, I., Kowalczyk, R., & Pham, H. H. (2002, January). Intelligent agents for automated one-to-many e-commerce negotiation. In *Australian Computer Science Communications*(Vol. 24, No. 1, pp. 197-204). Australian Computer Society, Inc.
Scacchi, W. (2002). Process models in software engineering. *Encyclopedia of software engineering*.

Schei, V., & Rognes, J. K. (2005). Small group negotiation: When members differ in motivational orientation. Small group research, 36(3), 289-320.

Schlin, B. (2019). What the f**k is MVC? Explained with easy analogy – Blake Schlin – Medium. Retrieved from https://medium.com/@blakeschlin/what-the-f-k-is-mvc-explained-with-real-world-analogy-e088a9a8b787

Schwartz, B., Zaitsev, P., & Tkachenko, V. (2012). *High performance MySQL: optimization, backups, and replication*. " O'Reilly Media, Inc.".

Srinivasan, S. S., Anderson, R., & Ponnavolu, K. (2002). Customer loyalty in e-commerce: an exploration of its antecedents and consequences. *Journal of retailing*, *78*(1), 41-50.

Su, S. Y., Huang, C., Hammer, J., Huang, Y., Li, H., Wang, L., ... & Lam, H. (2001). An internet-based negotiation server for e-commerce. *the VLDB Journal*, *10*(1), 72-90.

Wang, X., Shen, X., & Georganas, N. D. (2006, May). A fuzzy logic based intelligent negotiation agent (FINA) in ecommerce. In *Electrical and Computer Engineering, 2006. CCECE'06. Canadian Conference on* (pp. 276-279). IEEE.

Williams, H. E., & Lane, D. (2004). *Web Database Applications with PHP and MySQL: Building Effective Database-Driven Web Sites*. " O'Reilly Media, Inc.".

Yan, Z., Yan, F., & Fong, S. (2007, September). Incorporating Knowledge into e-Commerce Automated Negotiation. In *Database and Expert Systems Applications, 2007. DEXA'07. 18th International Workshop on* (pp. 600-604). IEEE.

Yang, Y., Singhal, S., & Xu, Y. C. (2009). Offer with choices and accept with delay: a win-win strategy model for agent based automated negotiation. ICIS 2009 Proceedings, 180.

Yu, L., Masabo, E., Tan, L., & He, M. (2008). Multi-Agent Automated Intelligent Shopping System (MAISS). The 9Th International Conference For Young Computer Scientists, 665-670. http://dx.doi.org/10.1109/icycs.2008.35