

Argupedia

6CCS3PRJ – Individual Project

April 2020

Author: A K M Naharul Hayat

Student Number: 1750583

K Number: K1764014

Supervisor: Dr. Sanjay Modgil

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. In addition, I confirm this report does not exceed 25,000 words.

A K M Naharul Hayat

April 2020

Acknowledgements

I would like to thank my supervisor, Dr. Sanjay Modgil, for providing me the right guidance to see this project through to the end. Without his support and supervision, it would not have been possible.

Abstract

Since the dawn of the information age, people have turned to debates online to understand both sides of an issue and subsequently, form a viewpoint based on their interpretation and comparison of the two sides. Majority of existing online debate platforms allow participants to engage freely in the form of text, without enforcing any structure. While this does have its advantages, it however, in practice, often results in debates where participants propose actions and make assertions without justification and rationalization behind them. This coupled with the fact that discussions can easily expand to a large number of pages, makes it difficult for new individuals looking to jump in and understand the different positions involved and whether or not the respective positions are regarded as accepted.

This project aims to build and evaluate an online application which aims addresses the issues by deconstructing typical structure of rational arguments in debates and researching ways we can incorporate it in the application to ensure debates remain focused. Secondly, it aims to visualize debates, so it is easy to keep track of its evolution, with the added benefit of the user not having to go through pages of texts. Finally, it focuses on exploring theoretical notions of classifying positions in debates and integrating it in the application to enable users to know at a glance which arguments are regarded as accepted at the current stage of the debate.

Contents

1. Introduction	8
1.1 Background and Context	8
1.2 Defining the Problem	8
1.3 Closer Look at Existing Solutions.....	9
1.4 Goals and Objectives.....	10
1.5 Conclusion	10
2. Background & Literature Review.....	10
2.1 Modelling Atomic Argument Structure	11
2.1.1 Walton's Argumentation Schemes.....	11
Appeal to Expert Opinion [19]	11
Argument from Position to Know [19]	12
Appeal to Popular Opinion [19]	12
Argument from Analogy [18]	12
Argument from correlation to cause [18]	13
2.1.2 An Extension of Walton's Scheme	13
Action Scheme	14
2.2 How argumentation scheme can be used to address the problem.....	14
2.3 Modelling debates using argumentation schemes	15
2.4 Labelling accepted arguments at a given state of a debate.....	16
2.4.1 Abstract argumentation theory of Dung.....	16
Foundation of dung's argumentation framework.....	16
Argument defence	17
Conflict Free Sets	17
Admissible Sets	17
2.4.2 Semantics of Acceptance (Extensions) - Dung's Argumentation Framework.....	18
Complete Extension	18
Grounded Extension	18
Preferred Extension	18
Stable Extension.....	18
2.4.3 Credulous and sceptical acceptance	19
2.5 Modelling debates as Dung's argumentation framework to facilitate labelling	19
2.5.1 Modelling using Argumentation Schemes	19
2.5.2 Modelling using textual entailment.....	21
2.5.3 Argumentation Schemes vs Textual entailment	21

2.6 Evaluation of Semantics of Acceptance (Extensions)	21
2.7 Grounded Extension Labelling algorithm.....	22
2.7.1 Pseudocode	23
3.0 Requirements.....	23
4.0 Specification.....	24
5.0 Design.....	26
5.1 Use Case	26
5.2 Project Solution Nature.....	28
5.3 Application Architecture & Justification	28
5.4 Database Schema & Entity Relationship Model	30
5.4.1 Justification of Database Design	31
5.5 State Machine	32
5.6 Important Note	32
6.0 Implementation	32
6.1 Development Methodology	33
6.2 Key Implementation Decisions & Their Justification	33
6.2.1 Development Framework & Technologies	33
6.2.2 Database Layer.....	34
Model Classes	34
How models were implemented to provide value	34
6.2.3 Presentation Layer (Front-end).....	35
Using template language to ensure front-end logic is easy to maintain	35
6.2.4 Routing Implementation.....	37
Benefits of having separate routing component	37
6.2.5 Controller Implementation	38
6.2.6 Styling Implementation.....	38
Styling framework	39
CSS Styling Unification	39
6.2.7 Modelling debates in implementation	39
6.3 Implementation of Requirements	40
R1 & R2 - Foundation of web development & front-end	40
R3 – Secure Login & Registration	40
R4 – Allow structured input to argument by user based on scheme chosen	40
R5 – Displaying created debate in the application	41
R6, R7, R8 & R9 – Critiquing and Countering Arguments	42
R10 – User has to be logged in before being able to engage in debate	43

R11 – Visualization of debate as graph	43
R12 – Displaying argument data on mouse-hover over node in graph	44
R13 – Jumping into sub-arguments by clicking on nodes on the graph	44
R14 – Marking initial debate argument in debate	44
R15 – Labelling algorithm and marking nodes in graph accordingly	45
R16 – Upvote/Downvote Arguments.....	46
R17 – Administration side of the application	46
6.4 External Libraries Used and Their Purpose	47
6.5 Instructions on running the project	48
6.5.1 Accessing admin site of the application.....	48
7.0 Testing.....	48
7.1 Testing Approach	48
7.2 Requirement Based Testing	49
8.0 Evaluation	51
8.1 Evaluation based on Nielsen’s ten usability Heuristic	51
8.1.1 Visibility of System Status	51
8.1.2 Match between system and the real world	52
8.1.3 User control and freedom.....	52
8.1.4 Consistency and standards	52
8.1.5 Error prevention.....	53
8.1.6 Recognition rather than recall	53
8.1.7 Flexibility and efficiency of use	53
8.1.8 Aesthetic and minimalist design	53
8.1.9 Help users recognize, diagnose, and recover from errors.....	54
8.1.10 Help and documentation	54
8.1.11 Conclusion and Reflection	54
8.2 Evaluation of Argupedia Debates	54
8.3 Evaluation of Visualization of Debates	55
8.4 Evaluation of Project.....	55
9.0 Legal, Social, Ethical and Professional Issues.....	56
10.0 Conclusion & Future Work.....	57
11.0 Bibliography	58

1. Introduction

1.1 Background and Context

Being one of the oldest internet services, online discussion forums to this day continue to stand out in terms of popularity. This is evident from the fact that popular online discussion platforms such as Reddit and Quora have over 300 million active user bases [1, 2] with the former ranking in top 20 in terms of popular sites on the internet [3].

The user generated content nature of such platforms allows its users to not only have discussions about a topic, but also have debates about pressing issues around the world and exchange idea and opinions by posting and replying to critique each other's comments/posts.

Debates at its core involve critiquing and presenting reasoning behind a proposed action or view, in order to justify it. Observing debates from this point of view enables us to consider its potential outside of mere online discussions. As noted by researchers in the field [4], debates can be evaluated and used by political leaders as a method for gathering public opinion and their reasoning - for and against a particular proposed policy or an issue. In addition, it has the potential to enable general public to observe a debate and have a well-informed educated opinion about a topic or an issue they lack knowledge about. Furthermore, it encourages them to analyse a proposed argument critically.

This project aims to build an online debate platform with the goal of realizing full potential of online debates as discussed above by addressing key issues with existing debate platforms, which we will discuss in the next section.

1.2 Defining the Problem

Having discussed all of the potential uses of debates in the previous section, there exists some issues which prevents the use of debates in existing online platforms to its full potential. These are discussed below:

1. It is often the case that debates end up having a lot of arguments. For example, one of the popular sub-forums in Reddit known as 'Change my View', which is strictly debate based, average close to 100 comments per topic [5]. In such long and complicated debates, it is difficult to make sense of which argument is critiquing which argument and reason through the debate and trying to extract the current accepted argument/position in a debate. As a result, making it difficult for individuals to have an informed opinion about a topic or an issue. This could discourage further participant from joining in on debates, because they may feel they do not know enough about the current state of the debate.
2. Existing debate platforms allow people to engage in debates by claiming a fact without backing it up with accepted ways of reasoning to reach the claim as the conclusion. This further adds to the difficulty of reasoning through a debate to a conclusion as it involves jumping through arguments which do not truly contribute to debate (i.e. arguments which are not backed by facts/reasoning/justification).

3. Existing online debate platforms focus on popularity of arguments/comments and favours them in terms of visibility, hence are generally considered as accepted argument in the debate. However, this has the potential of introducing bias in a debate. As noted by Scott in his book [7], people often tend to fall victim to confirmation bias, which is the tendency to favour information in a manner which tends to support one's previous belief. Participants in a particular forum, may be biased towards a particular area, therefore, only voting positively for arguments which confirm their bias and negatively to the ones which do not. This in turn diminishes the chance of arguments which are against the bias to be noticed by public, despite being perfectly valid. Therefore, individuals exploring a debate topic to form an informed opinion, may only notice the arguments which favour bias and interpret them as accepted. Furthermore, perfectly valid arguments may not get the recognition it deserves just because it was posted at an inopportune time (later in the debate) as a result failing to get votes.
4. Online debates do often times get a bit heated. In such situations, it is observed people often ridicule contributions proposed by opposition, thus negatively affecting the overall debate as well as the motivation of participants. A study in the field regarding experiences of people participating in online forum and communities [6] suggests that close to 4% of participants encounter bullying. Although the figure may seem quite small, however, it is worth noting that public forums are a community as a whole, this could have an impact on other participants indirectly and dissuade them from engaging in debates in a healthy manner. However, the impact of this is yet to be evaluated.

1.3 Closer Look at Existing Solutions

There however exists online platforms such as 'Debatepedia (www.debatepedia.com)' and 'Kialo (www.kialo.com)', which although less popular than the ones mentioned before, attempts to address some of the problems mentioned above. Both of these platforms groups arguments in the form of pros and cons which allows users to easily weigh both sides of an issue/topic. In addition to this, 'Debatepedia' allows users access to detailed background information about a topic. 'Kialo' on the other hand also allows users to dive into sub-debates regarding a main topic in order to enable user to focus on targeted discussion. Furthermore, it allows visual representation of a debate in the form of a tree, with the main initial argument being represented as the root node of the tree and arguments which support or attacks it, being represented as child nodes.

However, both of the platforms mentioned allows users to engage in debate without following an accepted reasoning pattern, which opens doors to users engaging in debate by simply stating their claims without reasoning behind how they are reached their conclusion. In addition, 'Kialo' for instance, judges the strongest argument and ranks visibility by user votes alone, which, as discussed previously, gives rise to problems such as low visibility of perfectly valid counter-arguments, due to the majority of members having bias themselves or due to the argument being posted very late in the debate leading to lower votes. Furthermore, the platforms do not have a clear concise

way of identifying arguments which are currently at an accepting position in the debate.

1.4 Goals and Objectives

As discussed in the previous section, majority of current online debating platforms require participants to engage openly in text form. Although this has its benefits in terms of freedom of input, however, in practice, it also leads to discussions where participants engage in debates without accepted forms of reasoning to justify their case. In essence, there is no structure being enforced which incentivize participants to engage in debates using accepted forms of justification and reasoning in everyday life.

The goal is to model typical structures of rational arguments which perfectly encapsulates what we consider as logical form of reasoning in everyday communication, and then, subsequently, enforce or incentivize users to engage in debates using the model.

The hypothesis is that doing so, will result in individual contributions in debates to follow what we consider as accepted form of reasoning. Hence will eliminate the problem of people engaging in debates without proper justification behind their claim. In addition, enforcing such a structure also have the potential of ensuring arguments in debates stay focused and eliminates the scope of the issue of online bullying and ridiculing in debates.

In addition, an exploration towards identifying 'accepted' positions of a given state of a debate is needed, however, it must not rely solely votes because of the potential of inflicting bias, as discussed previously. Having a method of classifying individual arguments in debates to identify which of the positions make logical sense will address the issue and make it easier for new participants looking to contribute. Furthermore, it will also ensure people can understand current state of a debate more easily.

1.5 Conclusion

Taking into account the points discussed in the previous sections, we can conclude that problems do exist in existing online debate platforms which prevents debates being useful to the full extent of their potential. The aim of this project is to research, develop and evaluate a solution which attempts to address the problems discussed.

The next chapter aims to discuss existing research in the area and evaluate whether it can be used to address the problem, if so, then in what ways we can potentially build on it to cater towards our problem and accomplish the goal discussed in previous section.

2. Background & Literature Review

This chapter intends to explore and discuss existing research in the area and evaluate whether it caters towards our goal and problem. If so, we aim to discuss how we can build on it to address the problems defined and whether there are other alternatives as well.

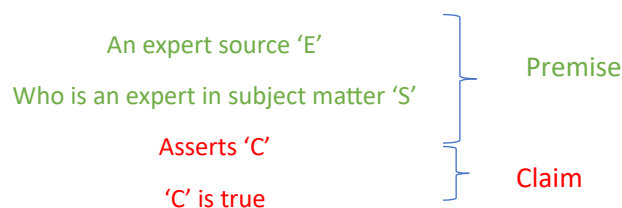
2.1 Modelling Atomic Argument Structure

In the initial part of the chapter we will dive deeper into existing research which models atomic arguments in debates, based on logical forms of accepted reasoning in everyday communication. The goal of this chapter is to understand ways we could structure individual arguments coming from both parties in a way such that it encourages justification of points being made.

2.1.1 Walton's Argumentation Schemes

Walton's Argumentation schemes [19] are structures of arguments which reflects how we reason in everyday conversation exchanges. It defines structure for common types of arguments we use in everyday conversation. The core inference consists of a set of premises followed by a conclusion/claim. Researchers in the area have put forward common argumentation schemes which are regarded as commonly accepted ways of reasoning to put forward a point in conversation. In addition to this, each argumentation scheme consists of its own set of critical questions [19], which reflects the way we normally think critically about a claim. An example of a common argumentation scheme known as 'Appeal to Expert Opinion' (taken from source [19] for demonstration) is as follows:

Appeal to Expert Opinion [19]



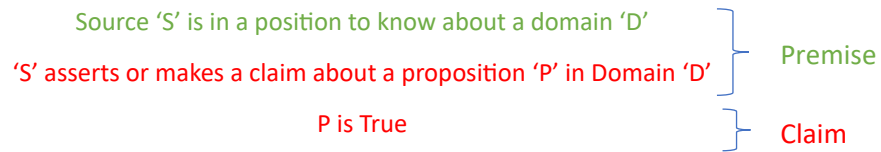
Note that the argument is rationally persuasive on the basis that there is a natural tendency to respect an expert but if we look at from inductive reasoning [20] or deductive reasoning perspective [20], it may not be valid, however that is the point of Argumentation schemes as it focus is to capture reasoning in everyday conversation exchanges.

As mentioned previously, each argumentation schemes consist of critical questions. A few example critical questions (taken from source [19]) for the above argumentation scheme is as follows:

- Is expert source really trustworthy?
- Can the expert 'E' really be considered an expert in the subject matter S?
- Does another expert make a claim which contradicts with C?

In addition to expert opinion, the following accepted argumentation scheme exists, each with their own structure (considered as rationally persuasive in everyday conversations) and critical questions:

Argument from Position to Know [19]

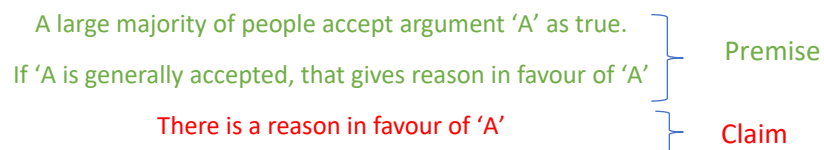


Note that the above argument reflects persuasion on the basis that if an entity is in a position where he possesses information about a domain, we normally assume that he is able to make claims which can be regarded as true. For example, a local in an area may be in a position to know about, for instance, nearest supermarket location. Again, such arguments may not have basis with regards to inductive and deductive reasoning, however, argumentation schemes focus on capturing reasoning in everyday communication exchanges.

A few example critical questions (taken from source [19]) for the above argumentation scheme is as follows:

- Is 'S' really in position to know in the domain?
- Is 'S' a reliable source?
- Did 'S' really assert that P is True or False?

Appeal to Popular Opinion [19]

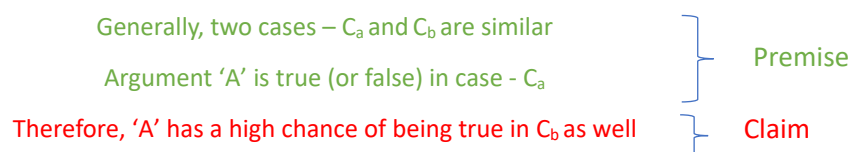


Note that the above argumentation scheme reflects persuasion on the basis that if a large majority believe in something, there must be a valid reason in the argument. It is worth noting that Walton himself considers this argument as deductively invalid, however, it perfectly captures the perception of an argument which is widely believed to be true in everyday life.

A few example critical questions (taken from source [18]) for the above argumentation scheme is as follows:

- What verification is there to support that 'A' is generally accepted?
- Does any other evidence go against 'A'?

Argument from Analogy [18]

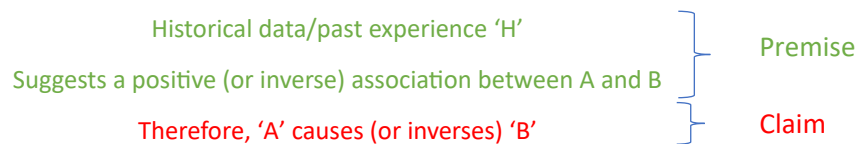


The above argumentation scheme reflects persuasion on the basis that if two situations are similar and an argument is true in one of them, then there is a high probability of it being true in the second situation as well. It reflects how we reason in everyday life based on existing examples which are similar to the one in question.

A few example critical questions (taken from source [18]) for the above argumentation scheme is as follows:

- In case C_a , does the argument 'A' really hold?
- Is case C_a and C_b , similar in the right way to assume 'A' will hold true in C_b as well?
- Is there a third case – C_c , which is similar to C_b and C_a in which the argument does not hold?

Argument from correlation to cause [18]



The above argumentation scheme reflects persuasion on the basis of data suggesting one thing leads to the other. It reflects out reasoning based on past experience and confirmation bias [7].

A few example critical questions (taken from source [18]) for the above argumentation scheme is as follows:

- Is there really a correlation between A and B? If so, was the method used to establish the relation valid?
- Is there any explanation which suggests that the correlation may have been a coincidence?
- If there a third element that triggers the correlation?

2.1.2 An Extension of Walton's Scheme

As pointed out by researchers in the field [21], the initial debate basis consisting of the proposal should start on a note which captures the following:

- A straightforward description of the rationale for a proposed action, which explicitly renders all the components of the logic in support.
- An opportunity to critically evaluate and question – not only the argument, but also relations between the logic behind it.
- A scope of proposing alternate solution or justification.

Although, the previous argumentation schemes discussed, manages to capture everyday reasoning and allows critiquing, however they fail to incorporate all of the points of the initial debate mentioned above.

Action Scheme

This leads us to an extension of Walton's argumentation scheme - known as 'Action Scheme' [22] which aims to capture the factors necessary for the initial argument of the debate. Its structure is as follows:



The action scheme (extended based on Walton's Schemes) enforces the participant to make the following points, which addresses the characteristics necessary for the initial argument:

- Encourages the person initiating the debate to demonstrate a knowledge of the present situation from his point of view.
- A description of the circumstance that would arise from executing the action.
- Characteristics of the new condition arising from the action which are considered to be appealing.
- The beneficial values which are promoted as a result.

Although the action scheme is particularly well suited for the initial debate argument as discussed. However, it can be used in counter arguments as well, to propose alternative solution, thus, the structure is not limited only to initial argument in the debate.

A few example critical questions (taken from source [21]) for the action scheme is as follows:

- Is the description of the current situation accurate?
- Will the action proposed really achieve the goal?
- Does the goal realize the value?
- Is the value really worth promoting?
- Will the action result have undesirable side effects?

2.2 How argumentation scheme can be used to address the problem

If we take a high-level view of argumentation schemes, they have the potential to be used as tools by debaters to make a claim/point. By basing their claim on an argumentation scheme structure, they conform their argument to ways of proposing arguments considered as persuasive and rational in every day communication. Hence both parties can propose their points in a debate by basing them on the structure defined by the schemes.

This leads to the debates being significantly more concise and rational, because each argument has to follow a clearly defined structure, which forces participants to focus on the point and also helps them follow proper accepted ways of reasoning to conclusion in exchanges.

Using argumentation scheme to enforce the use of structured arguments in debates, addresses the problem of online debates having responses which are baseless, or in other words, responses which critique without following proper reasoning/backing with fact, as participants are now forced to structure their argument using argumentation schemes. Moreover, this also addresses the problem of contributors ridiculing each other's argument as responses have to follow a structure.

2.3 Modelling debates using argumentation schemes

In the final part of the previous section, we discussed how argumentation schemes can be used to put forward individual points in the debate by participants.

To elaborate on the idea further; each scheme has their own set of critical questions which have the potential to encourage both parties to think critically about the proposed argument. Subsequently, critical questions can be used as a basis for critiquing the argument by making their own counter-argument, again using one of the argumentation scheme structures discussed. Enforcing the use of critical question as basis of proposing counter argument, helps participants focus on the exact aspect of the original argument he is attacking, which leads to criticisms which are focused.

To illustrate, suppose a participant starts a debate topic using the 'Action Scheme' and he proposes the initial argument as follows:

- **In Situation:** Coronavirus spreading around the country resulting in deaths
- **Doing action:** Imposing total lockdown in the country
- **Achieves Goal:** Reduce infection rate
- **Promotes value:** Good health

The argument proposed above is based on the action scheme and can be critiqued using one of the critical questions of action scheme. Suppose another participant comes along and criticizes the argument on the critical question – 'Action proposed have an adverse side effect'. Subsequently, he can form his own argument on the basis of the critical question. An example counter argument based on 'Expert opinion' scheme which critiques the argument above is demonstrated below:

Critique position: Action proposed could have adverse side effect

- **Expert Source:** Economic research department at 'Buzzfeed'
- **Expert Area:** Specializes in macro economics
- **Expert assertion:** Asserts that imposing lockdown will result in poverty, as sick leave wages proposed by government is not enough.

To expand the example a bit further, another participant may come along who disagrees with the expert opinion argument above and can critique similarly based on the critical question of expert opinion scheme.

Just to demonstrate an example, the new participant could critique the above argument using any argument scheme he/she chooses. For this instance, let us assume he chose to counter the argument using 'appeal to popular opinion':

Critique position: Expert source is questionable

- **Popular claim:** Large majority believe 'BuzzFeed' often reports misleading information
- **Assertion based on popular claim:** Research claims from BuzzFeed should be taken with a grain of salt.

Note that the examples above was purely based on fiction and is simply used to demonstrate how a debate based on argumentation schemes and critical questions - may evolve.

There does exist other argumentation schemes devised by researchers in the field. However, for the initial iteration of the project, the aim is to evaluate using the six schemes introduced in the previous section of this chapter.

2.4 Labelling accepted arguments at a given state of a debate

As defined in the problems, an exploration towards identifying accepted arguments/positions at a given state of a debate is needed, however, the method much not solely rely on votes, as it has the possibility of inflicting bias.

This brings us to studies in the field of 'Argumentation Theory which explores dealing with disputable information and applying logical reason in order to reach a conclusion [8]. Several scholars in the field of Artificial Intelligence and Philosophy have researched methods of argumentation and its position in everyday reasoning. This section attempts to explore current knowledge, including empirical observations, and conceptual approaches to labelling positions in debates.

2.4.1 Abstract argumentation theory of Dung

A lot of research on the subject of argumentation is focused on Dung's abstract argumentation frame-work [9]. It is abstract [10] in the sense that its core fundamental is only concerned with a set of atomic arguments coupled with a binary relation of the arguments representing which argument is criticizing/attacking which argument. It does not concern itself with the structure of the arguments or the core meaning behind how it is. However, it is widely used because it is very powerful in the sense that it can be used to deduce and identify set of logically accepted arguments from a knowledge base consisting of simply atomic arguments coupled with binary relation of their attack/criticism relation.

Foundation of dung's argumentation framework

The concept of argumentation framework can be represented visually [11] in the form of a directed graph, with the nodes, depicting the atomic arguments, and the binary attack/criticism relation set being represented as arrows between the nodes. This is represented in an example, inspired by [13], demonstrated below:

Atomic argument A1: The cheapest option is to go to the park; therefore, we should go to the park tomorrow.

Atomic argument A2: But it is cold tomorrow, therefore we should go to café instead.

Atomic argument A3: Forecast says it will be hot tomorrow, so you won't be cold tomorrow.

This can be represented visually as follows:



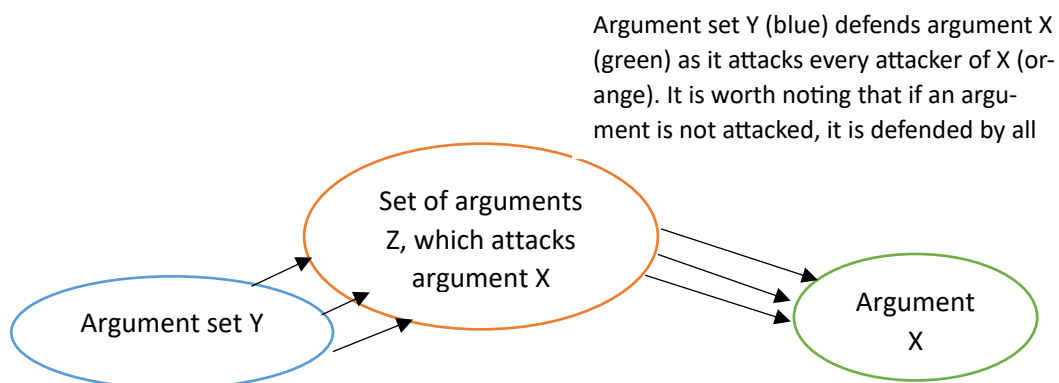
Node that an arrow from an argument node, say for instance X to an argument node Y conveys that, X is attacking argument Y [11]. In the example above, A1 is attacking A2 and A2 is attacking back as well. This is because A2 is criticizing A1 [13], backed by the reasoning that “it is going to be cold tomorrow”, how-ever, A1 is attacking back as well because it is claiming the opposite of A2, and is also backed by the rea-soning that “it is the cheapest option”. Note that A3 is attacking A2 on its claim that it is going to be cold, backed by reasoning of weather forecast, however, A2 is not attacking back [13] in this case because its claim – ‘it is going to be cold tomorrow’, which is being attacked, is not backed by any reasoning. Hence in this case it is a one directional arrow.

The argumentation framework for the example above can be formally defined as:

Atomic Arguments = {A1, A2, A3}, Attack Relation = {(A1, A2), (A2, A1), (A3, A2)}.

Argument defence

The concept of argument defence is illustrated below [13]:



Conflict Free Sets

An argument set is said to be conflict free [11] if none of the arguments in the set attack each other. I.e.: they are not conflicting. For instance argument set {a, b, c} is considered conflict free if there are no arrows between a, b, or c (no attack between each other each other or no bad blood in the set of arguments).

Admissible Sets

In essence, admissible sets [14] are sets of argument subsets, which are self-defended. In order to compute admissible sets, we take the following steps:

1. We take the power set of all arguments. For example, if we had only arguments A and B in the graph, the powerset would be: {(a), (b), (a, b)}.
2. Once we have the powerset, we eliminate the subsets in the powerset which have conflict, and only keep the ones which are conflict free. In the example

above, if there is an arrow from $a \rightarrow b$ (a attacks b), as in the set (a, b) is not conflict free, we eliminate it from the set of powerset.

3. For each subset in the filtered set in step 2:
 - If an argument in the subset is attacked, it must be defended by the arguments in the subset itself for it to be classified as admissible.

2.4.2 Semantics of Acceptance (Extensions)- Dung's Argumentation Framework

Given a visualization similar to the example demonstrated previously, one can then investigate whether an argument is to be regarded as accepted, also known as – 'Extensions' [12].

In essence the concept is that, an argument is regarded as accepted, if it can be successfully defended from all arguments which challenge/attack it [14], however, there are numerous further notions, which extends this idea and narrows it down, and each has its own definition of acceptability of arguments. These are known as 'Extension Based Semantics'.

To summarize, we have an array of possible means (Extension based semantics) of defining the type of logic to classify arguments in the framework as accepting. Each extension-based semantics have its own respective arguments which it deems as accepting. In the following section, we will explore different extension-based semantics we can use in Dung's abstract argumentation framework:

Complete Extension

Complete extension [9] extends the idea of admissible sets by imposing restrictions on it to strengthen it. A given set of argument is regarded as complete extension [9], if it is admissible and the set includes all arguments it defends.

A simple algorithm to compute complete extension as follows:

- For each admissible set
 - For each argument in the set
 - We find out if the arguments in the set defend anything.
 - If it does, it must be included in the admissible set itself for the set to be regarded as complete extension.

Grounded Extension

The idea of grounded extension [9] extends from complete extension semantics. Grounded extension [9] is the minimal-subset of complete extension. It is worth noting that the grounded extension can be empty set as well.

Preferred Extension

The idea of preferred extension [9] also extends from complete extension semantics. Preferred extension [9] is the maximal-subset of complete extension.

Stable Extension

The idea of stable extension [9] extends from preferred extension semantics. An algorithm based on intuition to compute stable extension as follows:

- For each set-in preferred extension
 - We find out which arguments the arguments in the set attacks
 - The set is considered as stable extension if it attacks all other arguments in the graph. (Excluding arguments in the set itself).

2.4.3 Credulous and sceptical acceptance

Each of the four extension-based semantics discussed above can be further broken down into either credulous acceptance or sceptical acceptance [12]. Credulous acceptance is more in sync with being lenient in accepting arguments and used in situation where the deciding agent have trust in agent who is putting forward the arguments. Sceptical acceptance on the other hand is the opposite and is stricter with regards to what arguments it accepts.

The extensions discussed previously can consist of multiple subset of arguments regarded by the extension as accepted. Among those subsets, the credulous extension consists of arguments which appear [13] at least once in any of the subset. On the other hand, for an argument to be sceptically accepted in the extension, it must be a part of all of the subsets.

2.5 Modelling debates as Dung's argumentation framework to facilitate labelling

If we abstract debates by disregarding supporting critiques and assuming counter arguments to a point always follow proper accepted ways of reasoning, then debates in essence, are just a set of atomic arguments with a binary attacking relation between them representing which argument is attacking which. This abstraction coupled with the assumption - results in a model which fits quite well with Dung's abstract argumentation framework.

The idea is that we abstract a debate and model it in the form of Dung's abstract argumentation framework as discussed. We are then able to visualize it in the form of directed graph. Once we have the visualization, we could use one of the extension methods of Dung's abstract argumentation framework to classify which arguments are regarded as 'accepted' in the debate. This addresses problem 1 and 3 defined in the first chapter.

2.5.1 Modelling using Argumentation Schemes

As mentioned previously, Dung's argumentation model requires the following:

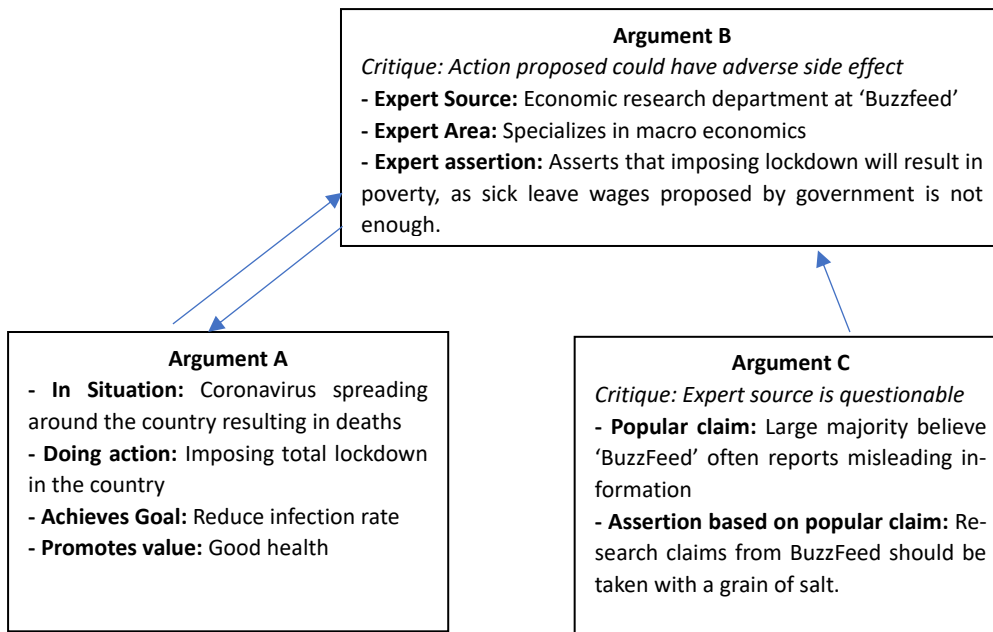
1. Set of atomic arguments
2. Their respective attack relation as input

In section 2.1, we introduced argumentation schemes which enforce exchanges focused on a single point only, thus resulting in arguments which are atomic. As mentioned above, atomic arguments perfectly fit input of Dung's argumentation framework model. Hence, for a particular debate, set of atomic arguments in the debate could simply be used as input in Dung's argumentation framework.

In section 2.1, we also introduced how critical questions can be used as a basis for critiquing a particular argument, which can be attacked subsequently, by forming a counter argument using another argumentation scheme. By extracting which atomic argument critiques and attacks which in the debate, we are able to extract binary relations of attack, which fulfils the second input mentioned above.

In addition, we are also able to derive whether the attack relation is unidirectional or bi-directional by referring to the critical question used as a critique and evaluating whether the counter argument is critiquing the premise or the claim.

For illustration, the example debate in section 2.2, modelled as Dung's abstract argumentation framework, is demonstrated below:



Atomic Arguments = {A, B, C}, Attack Relation = {(B, A), (A, B), (C, B)}.

Since argumentation schemes enforce exchanges focused on a single point only, thus the individual arguments are atomic, and this is reflected in the 'Atomic arguments' set in the framework above.

Note that the attack relations are derived based on the critique. For instance, the attack relation from argument B to A is based on the critical question 'Action proposed could have adverse side effect'. This critical question is critiquing the action proposed by A. Thus, B is critiquing a 'claim' made by A. B's attack is backed by its own set of premises, however, A's claim is also backed by its own premise. Hence the attack relation is bidirectional.

However, the attack relation between C and B is unidirectional. This is because C is critiquing the source of the expert in B, thus its critiquing the premise and not the actual claim made by B. C's attack is backed by its premises, however, the premise of B (expert source), is not backed by any of its premise, hence its a unidirectional attack relation.

In conclusion, debates structured by argumentation schemes can indeed be modelled as Dung's argumentation framework. Once we have it modelled as such, we are able to visualize and subsequently apply one of the extension methods discussed in section – 2.1.2 to label arguments regarded as 'accepted' at a point in debate.

2.5.2 Modelling using textual entailment

As mentioned in section 2.5.1, Dung's argumentation framework requires set of atomic arguments as its input.

We have discussed structuring debates using argumentation schemes and modelling it as dung's argumentation framework, however, researchers have approached modelling online debates in natural language in the form of dung's argumentation framework. This was done through making use of textual entailment [16] in natural language processing to deduce claims in conversations and inter-connection between claims/arguments.

Researchers approached [17] the problem by the assumption that natural language conversation in debate can be broken down into 2 types of snippet - one being the text and the other being the hypothesis. They have used textual entailment on the 2 snippets of the text to deduce if the meaning of the hypothesis can indeed be derived from the meaning of the text. If so, then the overall text input can be classified as an atomic argument.

They have also evaluated [17] accuracy of textual entailment system by measuring how well it is able to assign correct entailment relations on datasets extracted from 'Debatepedia' (www.debatepedia.com). Having used a short dataset of 100 pairs for both training and test, they have noted an accuracy of 67% [17] on the latter, however its performance on larger dataset is yet to be evaluated.

Once we have atomic arguments in debates using either textual entailment or through enforcement of argumentation scheme, the attack relations can be extracted based on the which original argument the replies in debates target. Finally, based on the two, we can model according to Dung's abstract argumentation framework and identify accepted arguments.

2.5.3 Argumentation Schemes vs Textual entailment

Having discussed both approaches, we can observe textual entailment have advantage with regards to user freedom of input, however argumentation schemes have the edge in terms of keeping arguments concise, focused and in alignment with structures which are deemed to follow the inferential reasoning we tend to apply in everyday life.

In conclusion, based on the discussion, argumentation schemes have the potential to better address the problems numbered 1 and 4 discussed in the first chapter.

2.6 Evaluation of Semantics of Acceptance (Extensions)

It is worth noting that the evaluating accepting arguments through Dung's abstract argumentation framework is considered as non-monotonic, in the sense that a new argument proposed in the future at a certain time, may lead to the previously accepted argument being no longer regarded as accepting [10].

Having considered that, there are numerous extensions that we could implement to label arguments in debate. This arises the question about what kind of semantics of acceptance or extension, among the ones discussed would be most suitable for the purpose? In this section, we will evaluate the extensions introduced in section - 2.4.2 and decide which extension would be most suitable for this application.

As pointed out by researchers in the field [15], the presence of a Preferred and Grounded extension for an argumentation framework is guaranteed, however for the case of Stable extension, there is no such assurance. In addition, there exists [15] precisely one grounded extension for an argumentation framework. On the other hand, there may be more than one [15] in Preferred and complete extension.

This makes grounded extension a good candidate for the application, as firstly, it extends from complete extension and is a refined version of it, secondly, we can have guarantee that at any point in debate, we have exactly one a set of arguments (which could be empty as well), regarded as accepted. Having multiple would bring us back to the problem at hand where people find it confusing and difficult to extract the current accepted position in debate.

The time complexity of the extensions, taken from source - [10] is as follows:

Complexity					
	<i>stable</i>	<i>adm</i>	<i>pref</i>	<i>comp</i>	<i>ground</i>
Cred	NP-c	NP-c	NP-c	NP-c	in P
Skept	coNP-c	(trivial)	$\sim P_2^P$ -c	in P	in P

[Dimopoulos & Torres 96; Dunne & Bench-Capon 02; Coste-Marquis *et al.* 05]

Source: [10]

As noted above, both credulous and sceptical grounded extension have a polynomial time complexity, which is better in comparison to the other three extensions being considered.

In conclusion, based on the discussion, the aim is to implement grounded extension to identify accepted arguments in debates for now, however, in the future, the plan is to have options for the other extensions as well.

2.7 Grounded Extension Labelling algorithm

Having reached the conclusion that grounded extension would be most suitable for the initial development. In this section, we will dive into the actual pseudo code implementation of the algorithm devised by researchers in the field.

2.7.1 Pseudocode

The algorithm for the grounded **extension** algorithm is adapted from source – [15]. The following is the pseudo code representation of the algorithm:

Input – set of arguments (A) in debate, Attack relation (R) between the arguments

Output – Sets containing arguments labelled as IN, OUT and UNDECIDABLE

```
Set IN
Set OUT
Set UNDEC

While True
  For each 'Argument' in debate which is not already in set – IN and Out
    If all attacker of 'Argument' is in Set OUT
      Append 'Argument' to set IN
    If at least one attacker of 'Argument' is in Set IN
      Append 'Argument' to set OUT
    If there has NOT been a change in set IN or OUT
      Break
  For each 'Argument' in debate which is not already in set – IN and Out
    Append 'Argument' to set UNDEC
```

3.0 Requirements

Following on from the literature review in the previous chapter, the core requirements of the project devised based on the analysis and review coupled with supervisor's recommendations, are as follows:

- R1. Based on the nature of the project, a web application solution fits well as it allows people all around the world to engage.
- R2. In addition, the website should be able flexible and saleable enough to be used on mobile phone browsers as well.
- R3. The application should allow users to register and login to their account.
- R4. Once the user is logged in, he should be able to create a debate topic by inputting a title and proposing an initial argument using one of the six Walton's argumentation schemes discussed in the previous chapter.
- R5. Once a debate topic is created, should be visible to all other users of the website.
- R6. In addition, they should be able to critique the argument by proposing a counter argument, however, before doing so, he will have to base it on the critical questions of the argumentation scheme of the original argument being addressed.
- R7. Once the user chooses his critique point, he should be able to choose one of the 7 argumentation schemes to base his counter argument on. Once chosen, he should be able to input text conforming to the structure to make his argument.
- R8. Once a counter argument is made using one of the six argumentation schemes discussed. All users of the website should be able to see the counter argument

proposed, displayed in the same structure as the argumentation scheme chosen.

- R9. Users should have to opportunity to critique the counter arguments in the same way as well. All users of the application should be able to view debate topics alongside all arguments/counter-arguments.
- R10. However, in order to engage in debates, they have to login using their credentials.
- R11. Users should have the ability to view visual representation of debate as dung's abstract argumentation framework, in the form of directed graph. Nodes should represent the arguments in debates and arrows among them should represent attack relation, for instance, if argument A is attacking argument B, the graph should have two nodes representing A and B, with an arrow from A to B representing the attack.
 - It may be the case that when an argument attacks another, the argument attacks back as the claim being attacked is already backed by its own premises. In this case, it should be represented in the graph as bidirectional arrow between the two nodes.
- R12. In the visualization, users should be able to hover mouse over an argument node in the visualization to view its respective content.
- R13. Users should be able to jump to a sub argument in debates and subsequently critique by clicking on nodes representing arguments in the visualization.
- R14. The node representing initial argument should be marked clearly in the visualization.
- R15. At any point in debate, set of arguments to be regarded as accepted according to grounded extension labelling algorithm should be computed and the respective argument nodes in the graph visualization, should be clearly marked for the user to see. The nodes should be clearly marked to represent which arguments are 'In', 'Out' and 'Undecidable'.
- R16. Users should also have the option to upvote/downvote an argument to represent its popularity. The user should be able to see votes for an argument when he hovers over the node representing it in the graph visualization.
- R17. The web application should have an admin side, through which argumentation schemes can be added/deleted/edited in the future. The admin side should also allow CRUD (Create, Read, Update, Delete) operations on the database.

4.0 Specification

In the previous chapter, we had defined requirements for the project. In this chapter, we will discuss specification for each of the requirements defined.

Requirement	Specification
R1	- Use a suitable web-based framework to streamline development and a clear and coherent structure across the project.

R2	<ul style="list-style-type: none"> - Use a suitable front-end framework which is scalable and caters towards both, websites viewed on mobile browsers as well as traditional browsers used on PC.
R3	<ul style="list-style-type: none"> - Use framework's built in security and authentication libraries to implement secure user registration and login
R4	<ul style="list-style-type: none"> - Structure of each argumentation scheme must be stored in an SQLite database for now. - The application has to be built in a way which allows flexibility in database engine, without requiring unreasonable amount of code changes. Hence, the schema of the database is to be abstracted using Model classes, so that the actual data retrieval code is independent of underlying database engine. - When the user attempts to create a new debate topic, the structure of the argumentation scheme chosen is to be retrieved from the database. Subsequently, appropriate number of text field conforming to the argumentation structure is to be displayed, with appropriate header under the field, so that the user can input the respective points he wants to make in the debate - Finally, appropriate form validation is to be configured to prevent invalid data submission.
R5	<ul style="list-style-type: none"> - Once the user inputs data of the initial argument, it is to be converted to a html format highlighting structure sections along with its respective content and subsequently, stored in database to be displayed in a coherent manner to other users, along with date/time of post and number of upvotes. - The initial argument is to be stored in database as a root node of a tree.
R6	<ul style="list-style-type: none"> - Other users seeing the initial argument of the topic, should have the option to critique the argument. - If he chooses to critique, we retrieve the respective set of critical questions of the argument which is being critiqued, from the database, and we allow the user to choose one the critical questions to base his counter argument on.
R7	<ul style="list-style-type: none"> - Once the user selects the critical question, he is to be shown a list of argumentation scheme options (retrieved from the database) to base his counter argument. - Once the user chooses an argumentation scheme for his counter argument, the structure of the respective argumentation scheme is retrieved from the database. Subsequently, appropriate number of text input fields corresponding to the scheme will be displayed along with their respective header, to allow the user to input his argument.
R8	<ul style="list-style-type: none"> - Once the user inputs his counter argument according to the structure, the content is to be converted to appropriate html format to highlight the structure headings. Subsequently, the argument is to be stored in database as child node of the argument that it is critiquing. Once database is updated, it should reflect in the front-end

R9	<ul style="list-style-type: none"> - The counter argument is to be displayed in a text-based tree structure for others to see on the website, with indentation to make it clear which argument is attacking which. Subsequently, other users should be able to attack the new counter argument created. The specification for this is similar to the counter argument creation mentioned in R6 and R7
R10	<ul style="list-style-type: none"> - Use framework's built in authentication system to restrict permissions according to user type. (anonymous or logged in)
R11, R12, R13, R14	<ul style="list-style-type: none"> - Using the tree representation of the arguments in the database for a particular debate, a JSON formatted text is to be constructed, which specifies the nodes, arrows, mouse hover text and click action for the directed graph. - Once the JSON format representing the graph is available, a graph visualization library compatible with the framework, is to be used to construct the visualization. - Event listeners are to be implemented in nodes to enable users to jump in, on the sub argument upon clicking a node. - Tooltips are to be implemented to display to the user the content of the argument along with its votes upon mouse hover.
R15	<ul style="list-style-type: none"> - Implement algorithm to extract grounded extension given data structured in a directed graph form. - Once the labelled arguments are identified, the JSON formatted data containing specification for the graph is to be amended and the nodes according to the labelling algorithm output is to be highlighted clearly in the graph.
R16	<ul style="list-style-type: none"> - An upvote/downvote option is to be available for the user to vote any argument. If one chooses to upvote/downvote, we simply add the user id and the respective argument id in the database, to mark the argument as upvoted/downvoted for the user. - The Upvote and downvote functionality is to be implemented with the help of Ajax, so the user can instantly view the updated vote count without the need to reload or refresh the page
R17	Admin site management functionality of the web development framework is to be used to automatically build the admin side of the website, which will allow administrator to modify schemes and their structure.

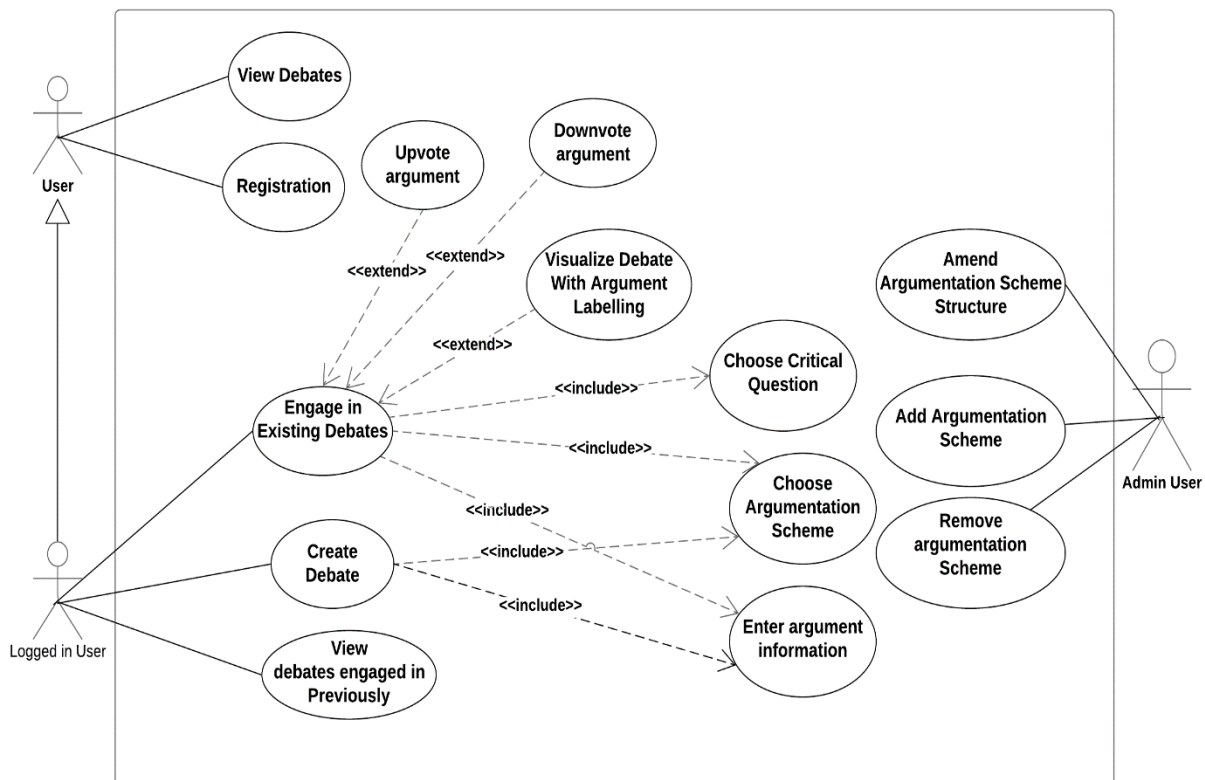
5.0 Design

This chapter aims to provide comprehensive description of design of the system, with the help of relevant diagrams.

5.1 Use Case

In system design, aim of a use case diagram is to illustrate the functionalities that stakeholders expect to derive from the system. From the perspective of developers, it adds value by being a point of reference to focus on the exact functionalities to be implemented. The figure below demonstrates the 'use case' diagram of the project:

Argupedia - Use Case Diagram



The main actor of the use case is the user of the system itself and the administrator of the system as illustrated in the diagram above. Detailed description of the use cases of the system are given below:

The actor 'User' represents users who are not logged in, on the system. They are still able to view and explore existing debates and view their content. In addition, they are able to register themselves onto the system.

The actor "Logged in User" inherits from the 'User' actor. As the name implies, this represents users who are logged in, on the system. The inheritance relation denotes the fact that logged in users are able to do the same tasks as the 'User' actor, with added use cases.

Users logged on the system is able to view existing debates with the added ability to engage. They are able to upvote/downvote existing arguments. In addition, they are able to visualize the debate and view arguments labelled by the algorithm on the graph. Moreover, they are able to engage by critiquing existing arguments by basing their counter argument on a critical question. Furthermore, they can also create their own debate topic by providing an initial argument. Finally, they are able to view a list of existing debate topics that they had engaged previously.

On the other hand, we have the actor labelled as 'Admin user'. As the name implies, this actor is the administrator of the system and has the privilege of accessing the administration functionalities of the system. Once logged in as administrator, he is able to change argumentation scheme structure, as well as add/remove them.

5.2 Project Solution Nature

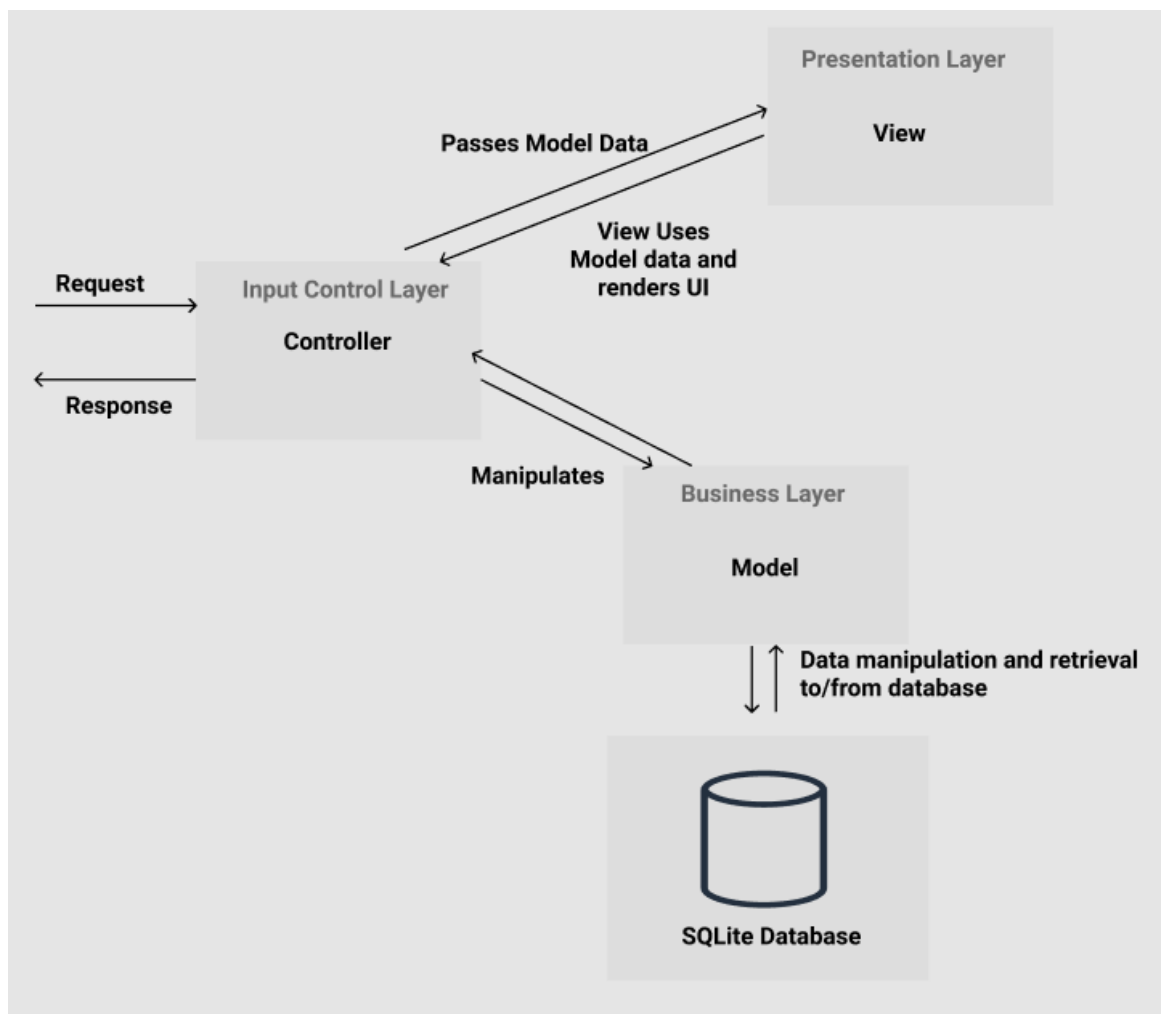
A web application was designed for the project as opposed to a desktop application/mobile application nature. The justification for this decision is outlined below:

- A web application allows greater reach in the community. It can be used by any device with a browser. Whereas, a mobile application is limited to specific mobile OS platform, such as IOS or Android. A web application can be used by a phone, as well as a laptop/desktop, as it only requires a browser.
- An application such as this, is expected to require further updates in the future. A web application solution is perfect for this, as it simply requires the developer to update the web-site hosted on the server, with minimal inconvenience for the end user. It is difficult to push updates on mobile application, as the user may have automatic updates disabled in their respective application store.

5.3 Application Architecture & Justification

In this chapter, we will discuss the architecture chosen for the application and its justification.

Model-View-Controller (MVC) architectural pattern was chosen for the project. The primary reason for the decision was due to MVC offering excellent separability of front-end and back-end components of the system. This pattern aligns with the



application's aim of optimising maintainability and reusability aspects. The following diagram provides an overview of the architecture:

As the architecture name implies, it consists of 3 distinct components, namely – Model, View and Controller. These are discussed below:

Model

Model is responsible for managing data of the application. It represents the application's dynamic data structure which is independent of the user interface. In essence, database interaction of the application is abstracted through the use of models and the benefits of doing so is outlined below:

- Having an abstraction through the use of model class ensures that the data manipulation logic of the application remains autonomous and independent of the database engine. Considering the fact that requirements of a software tend to constantly evolve, having flexibility of swapping database engine in the future without having to change query logic implementation is an advantage.
- Query logic relevant to a particular database table is encapsulated within the respective model class. For instance,
 - A model class responsible for abstracting a particular database table only includes query logic which is relevant specifically to the table. For example, suppose in the database schema, we have a table called 'Arguments', which is responsible for storing individual argument data in a debate. In implementation, this will be abstracted through the use of a model class, namely 'Argument'. Data manipulation logic of individual arguments – such as, retrieving upvotes, downvotes etc, will be contained and encapsulated in the 'Argument' class.
 - This also encourages reusability and prevents code duplication. If a particular query logic is needed elsewhere, we can just call the function in the relevant model to execute the logic. We do not have to implement it again.
 - In addition, models make it easier to follow through the data logic of the application. Suppose, we need to make a change to a particular data logic. It is easier to identify in which part of the application we need to make the change, as data logic for a specific table is located in its respective model.

Views

Views are responsible for the visual elements of the web application, which is presented to the end-user and it is completely independent of business logic. The advantages of a separate view component are outlined below:

- The front-end components (View) are isolated from every logic of the system including database query logic. This strengthens encapsulation and modularity of the application, aligning with quality software engineering practices.
- Having the separation enables parallel development with minimal friction. In addition, it enables developers to focus. For example, if an amendment is

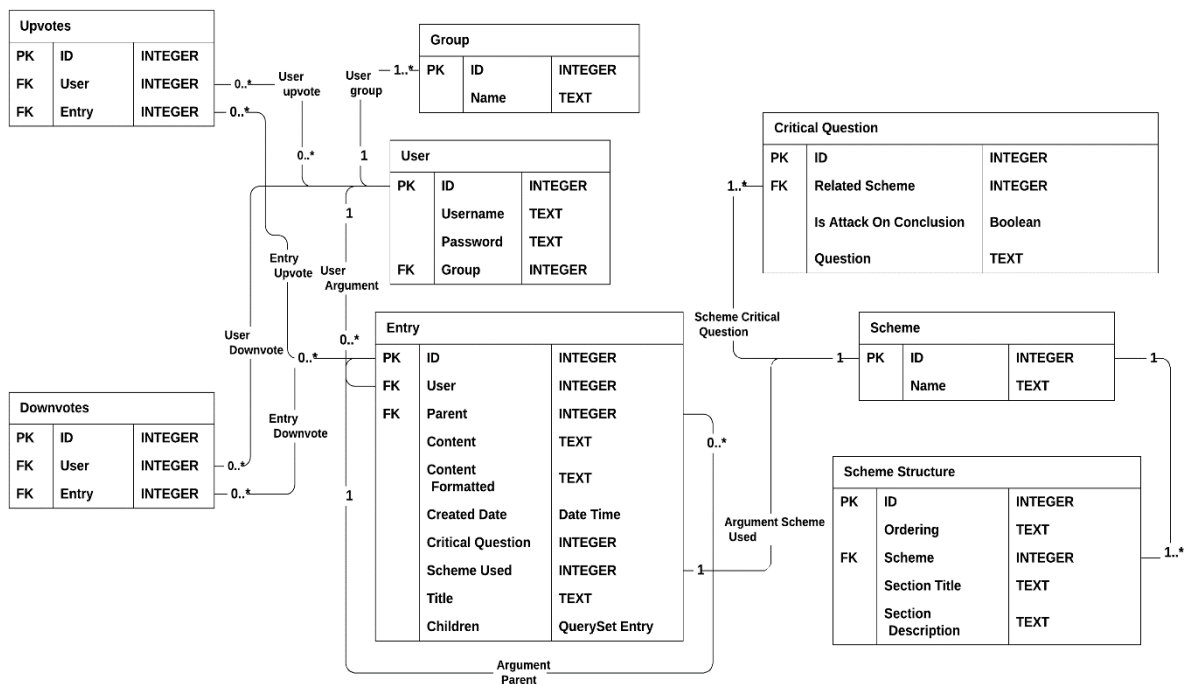
needed in the front-end of the application, developers is able to do so by solely focusing on front-end code and vice versa.

Controller

Controller is responsible for the complete orchestration of the view and the model. It is responsible for processing the request made by the client and subsequently provide response. Whenever client sends a request to the web application, the controller performs create, update, read and delete (CRUD) operations as required and passes model data onto the views, which in turn uses the model data to render the user interface as defined and returns response to the controller, which forwards response to the client via HTTP response.

5.4 Database Schema & Entity Relationship Model

Entity relational diagram is a valuable instrument for visualising the application's underlying database structure and relations between tables within. The schema was designed to minimize null values in database and according to third normal form. The diagram below demonstrates the schema of the database to be reflected using models:



The tables along with their relationship and cardinalities are described below:

- The 'Scheme' table contains data of the name of the argumentation schemes.
- The 'Scheme' table has a '1' to '1 or many' relation with the critical question table which holds critical question data. This relation denotes that each scheme must have at least 1 critical question
- The 'Scheme' table has a '1' to '1 or many' relation with the 'Scheme Structure' table which represents the headers of the scheme. For instance, the action scheme has headers – 'in situation', 'taking action', 'achieves goal' and 'promotes value', and so on and so

forth. The relation denotes that each scheme, must have at least 1 header. The 'Scheme Structure' table holds data such as ordering of scheme headers, title, description, etc.

- 'Group' table contains the different groups of users of the system, such as admin, anonymous, etc.
- 'User' table holds data of the user of the system, such as their username, password. it a one to many relationships with the Group table. This relationship classifies whether the user is an admin or not.
- The 'Entry' table contains data representing atomic arguments in the debate. Structure of arguments in debates are represented in the form of tree, where the initial argument is regarded as the parent, and its children its attacker and so on and so forth. In addition, each record in 'Entry' table has an association with a user, to record which user has made the argument. Furthermore, it has a recursive relation with itself. This is represented in the cardinality named - 'Argument Parent', as shown in the diagram above. The recursive relation is used to structure debates in a tree format, as discussed.
- The 'Upvote' table represents a 'many to many' relationship between 'Users' and 'Entries' (arguments). It has foreign key associations with 'Users' and 'Entries' table to record data about which user has upvoted which entry.
- The 'Downvote' table, is similar to the 'upvote' table. It represents associations to record data about which user has downvoted which entry (argument).

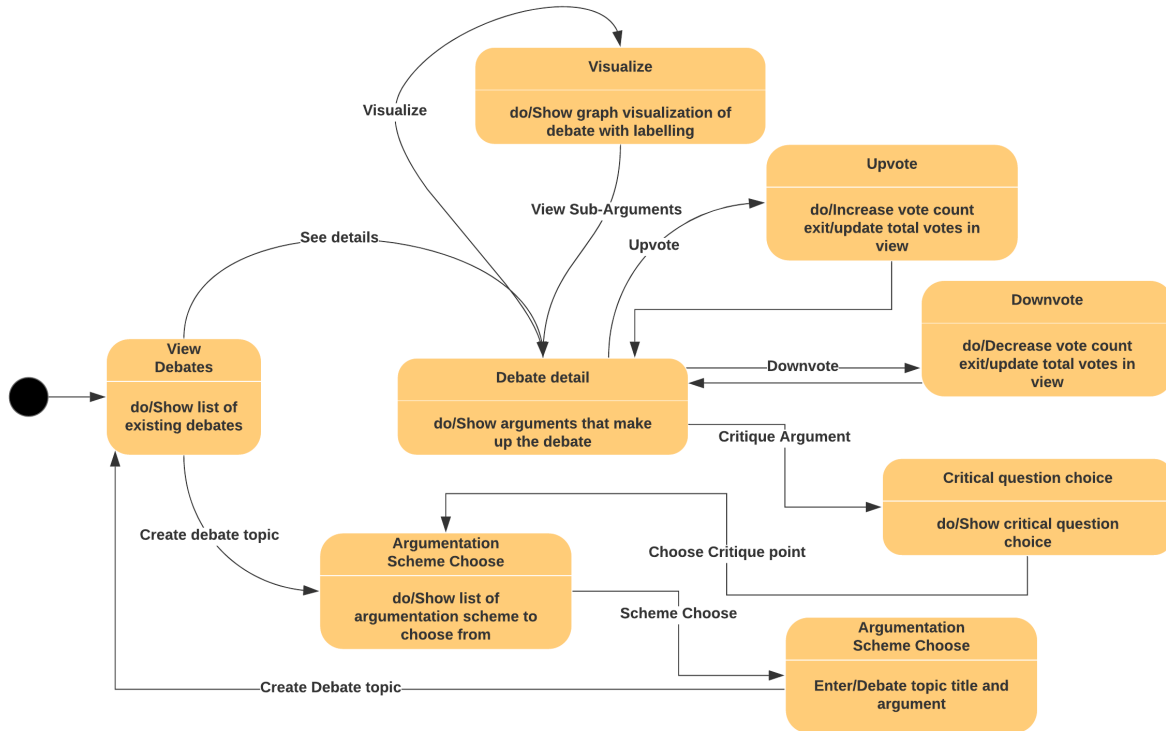
5.4.1 Justification of Database Design

The database has been normalized to third normal form where possible.

- Note that many to many relationships are separated into a different table, containing foreign keys of the two tables being linked. This is to minimize null values in database, which is considered good practice in database design.
- In addition, each table only consists of data field relevant to their respective entity. For example, the 'Group' table only consists of the group name. This is to ensure that the scheme is extendable and maintainable, thus adhering to good database design principles.
 - For instance, It could have been designed in a way such that 'Group' would be a field/column in the 'User' table, however, suppose in the future, we have to store more information about 'Groups'. If we take this approach, then this would not have been possible. Another similar example is the 'Scheme' table.

5.5 State Machine

State machine diagram reflects the flow of the application usage for the end user. It represents system's various states and the triggers that cause the transition from one state to the other. The state machine diagram of the application is shown below:



5.6 Important Note

Please note that an agile development methodology (elaborated in the next chapter) was used for the development of the project. Hence, system design diagrams such as 'Class diagram' and 'Sequence diagram', which are very specific to implementation, was not used in the design process.

The reason for as doing so was because - setting design of classes and their interaction, in stone, would have diminished the scope of flexibility and accommodating business requirement change, which was expected to be inevitable, considering this was the initial version of the application

Therefore, to accommodate for future business requirement change based on supervisor's feedback, design processes which are regarded as highly specific to implementation were not considered, at this early stage.

6.0 Implementation

In the previous chapter we discussed system design and key decisions made within along with respective justifications. This chapter aims to discuss how the design elements were incorporated and how the system was implemented overall. Initially, we will address key decision points in the implementation phase and subsequently, we will dive deeper into practical approach used to addressing the requirements and specifications defined previously.

6.1 Development Methodology

For the development of the project, Agile development methodology was chosen. Choosing an iterative development methodology for the project enabled having regular discussions with supervisor to ensure that the system being developed was consistent with his expectations. In addition, the preparation of different iterations of the project offered insight into important milestones of the project, which in turn enabled more efficient planning and understanding of scope. Moreover, it enabled more flexibility in accommodating changes in the application based on supervisor feedback.

6.2 Key Implementation Decisions & Their Justification

Applications are prone to even evolving demands. It is for this reason, maintainability and extendibility were among key factors - driving decisions made during implementation. In this section, we will discuss decisions made with regards to implementation of the project and their respective reasoning in support.

6.2.1 Development Framework & Technologies

As stated in the requirements chapter, a web application solution is more in alignment with the nature of the problem. In order to ensure efficient development of the project, decision was made to use 'Django', a python-based web development framework.

Django provides full set of benefits that come with using typical web development frameworks, such as efficiency, scalability and code-reusability. An overview of the justifications for choosing Django framework specifically for the development of this project is outlined below:

- Django provides authentication library which allows developers to implement industry standard security measures in the application. Some examples of essential security measures provided by Django is listed below:
 - Cookie encryption for sessions.
 - Cross Site request forgery (CSRF) attack protection.
 - Cross site scripting (XSS) protection.
 - SQL Injection protection
 - Host header validation
- It enforces Model View Controller architectural pattern (discussed in the Design chapter) which leads to excellent modularity of different components of the application.
- It abstracts database layer of the application through the use of Object Relational Mapping via Django Models, allowing flexibility in switching database engine in the future without major code changes in database logic, thus making the application database engine independent.
- It also allows easy and automatic management of admin side of the website along with automatic synchronization with changes in model, which gives Django an edge over other similar web application development frameworks in the industry.
- Provides error handling services for web application, including but not limited to displaying or appropriate error codes in case application runs into issue.

Considering the above-mentioned advantages of using Django framework, coupled with excellent community support of python as a programming language, the decision was made to use Django for the project.

6.2.2 Database Layer

The data layer of the application was implemented according to the architecture discussed in the design chapter. Underlying database engine used was Django's default - 'SQLite', however, the implementation was done using Django models, which allows implementation to be completely independent of database engine. This is illustrated below:

Model Classes

Model classes were used to represent the database schema, visualized by 'Entity relational diagram' in Design chapter. Models are defined in the 'models.py' file located at - '/Argupedia'. A code snippet of the project demonstrating model of the 'Entry' table in the schema, as an example, is shown below:

```
class Entry(MPTTModel):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="author")
    parent = TreeForeignKey("self", null=True, blank=True, on_delete=models.CASCADE)
    content = models.TextField(max_length=4000, default="", help_text="")
    content_formatted = models.TextField(default="")
    title = models.TextField(max_length=100, default="", help_text="", blank=True)
    upvotes = models.ManyToManyField(User, blank=True, related_name="upvotes")
    downvotes = models.ManyToManyField(User, blank=True, related_name="downvotes")
    created_date = models.DateTimeField(default=timezone.now)
    modified_date = models.DateTimeField(blank=True, null=True)
    scheme_used = models.ForeignKey(Scheme, on_delete=models.CASCADE, related_name="")
    critical_question = models.ForeignKey(CriticalQuestion, null=True, blank=True)
```

As demonstrated in the snapshot above, models define database fields of a table in a readable manner. For example, it is clear from the code snippet that the user foreign key relation has a 'cascading relation' and that the max length of title of an argument is 100 characters, as shown in the yellow highlighted area. In addition, foreign key relations are easy to understand at a glance as well. For example, in the 'red' underlined part in the snapshot above, we can easily know at a glance that the entry table has a field named 'downvote', which has a many to many relation with the 'User' model.

How models were implemented to provide value

As discussed in the previous section, the use of model provides representation of database in a simple, readable and easy to understand manner, which aligns with good software engineering principles. On the other hand, if raw SQL query was used to define the schema, then this would have been complete opposite. In our case, by simply looking at a model class, the developer is able to figure out almost everything about the entity. Furthermore, if change is needed at a database entity, it is easy to figure out where to make the change as logic related to each database entity is confined within the respective class.

Moreover, models play a significant role in reducing code duplication via methods implementing query operations. Query functions related to a particular entity in the database is confined/encapsulated within the respective model. the snapshot below shows an example method from the Entry (argument) model class, which calculates total votes of an entry (argument).

```
@cached_property
def votes_sum(self):
    """
    Returns subtract of downvotes from upvotes
    """
    return self.upvotes.count() - self.downvotes.count()
```

Note that the operation carried out by the method shown above (calculates total vote count for an argument) relates to argument entity, hence it is encapsulated within the Entry (Argument) model. In addition, the method shown in the example above can be called within any argument (entry) object to retrieve its total vote count. Therefore, the implementation promotes reusability which aligns well with software engineering principles.

Furthermore, Django's built in object relational mapper (ORM) is used to interact with the database models. This allows us to perform data retrieval tasks without having to write complex SQL queries. An example code snippet demonstrating use of ORM is shown below:

```
SchemeStructure.objects.filter(scheme=pk_scheme).order_by('ordering')
```

The code snippet above retrieves structure of the scheme being passed in and orders them. It is worth noting that the code is significantly more concise and self-explanatory, when compared to raw SQL queries.

6.2.3 Presentation Layer (Front-end)

The front-end implementation of the application reflects the architecture diagram demonstrated in the design chapter. Templates/Visual-elements (HTML files) to be presented to the end user are confined within 'templates' package in the project and are completely separate from all business logic of the application.

Using template language to ensure front-end logic is easy to maintain

Django template language was used for implementing logic for the front-end components. Using template language ensures that there is no mixture between HTML and raw python code used for business logic. As a result, it allows separation between front and back end components. Finally, it allows implementation of front-end logic by writing code which is clear, concise and significantly for readable, thus making the logic easier to maintain in the long run. This is demonstrated in the example below:

```

<div class="row ml-2" >
  {% if not user.is_authenticated %}
    <h7 class="pl-3"><a href="/accounts/login/">Login</a> or
  {% endif %}
</div>

```

Note that the front-end code logic (highlighted in yellow), is self-explanatory and very easy to follow and understand. As a result, developers revisiting code in the future would require significantly less effort to get an overview of the logic.

Cohesive directory structure of front-end components

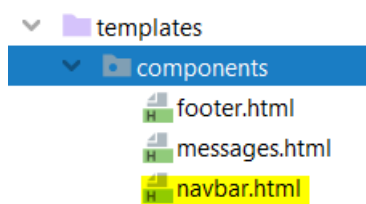
Front-end components of the application were into separate folders, each containing templates representing their own respective area. This is to make the directory structure more cohesive. This is directory structure of templates are demonstrated below:

- ‘/templates’
 - Contains front-end templates relevant for the general application, e.g.: Authentication templates such as login/registration page
- ‘/argupedia/templates’
 - Contains front-end templates specific to ‘Argupedia’ application, for instance, graph visualization page.

In the future, it is expected that as the application expands its feature set, templates will be further divided into more directory structure according to area of the application.

Preventing code duplication in front-end using template engine

Code snippets in the front-end which have high reusability scope are split into separate files, so that in the future, if we have to use it elsewhere, it can simply be included in the template, without having to reimplement it. This eliminates code duplication in front-end components of the application. Reusable code snippet of front-end are located in the directory – ‘templates/components’. This is further illustrated below:



As shown on the snapshot on the left, the directory contains the file ‘navbar.html’. This file only contains code concerning the navigation bar of the web pages.

If we have to change the navigation bar, we can just make the changes here and it would affect the entire application. This makes the code more maintainable and extendable and if changes are needed, we only need to change confined in a single file.

Improving maintainability in front-end layouts

Furthermore, a single cohesive layout has been set up for web-pages across the application. The front-end has been set up, such that it uses the same layout for consistency. This is further illustrated below:

The 'base.html' file (shown on the left) defines a consistent layout of webpages

The snapshot on the left, shows content of the layout file.

Note that it includes code which sets up the HTML tag layout of the page. If this layout is included in the pages, then we do not have to duplicate the code already written.

It also includes the navbar component (code snippet discussed previously), so it reflects across all pages

Finally, the snapshot below demonstrates the layout being included in a template file (HTML). Doing so, eliminate the need to set up the layout of the page again. This not only minimizes code rewriting, but also makes it easier to maintain - as if change in layout is needed, we only make to make changes at a single point.

```
{% extends 'base.html' %}

{% block title %}Sign Up{% endblock %}
```

6.2.4 Routing Implementation

To improve code cohesiveness and adhere to strong software development standards, Django's routing component was used to process URL requests. The routing component is located at – '/argupedia/urls.py'

The routing component match URL of request made by the user's browser and forward to appropriate function in controller, depending on whether the request is of type – HTTP GET or HTTP POST.


Benefits of having separate routing component

The following are the key driving decisions behind having a separate routing component:


- Routes provide a single point of contact for URL requests and maps them to controllers. If the single point of contact did not exist and URL mappings were scattered throughout different parts of the application, then it would be very challenging to maintain. Thus, having routes package improves maintainability.
- Routes also define valid URLs of the website. As a result, developers can simply have a look at the routing component to an overview of URL's considered as valid for the website. This also helps debug during development.

An example code snippet from the routing component is illustrated below:


```
path("posts/visualize/<int:pk_post>/", VisualizeView.as_view(), name="visualize-view"),
```




Valid URL and its parameter type defined



Controller class on which too route request



Method to call in controller class to handle the request



Name of the route

6.2.5 Controller Implementation

As quoted [23] in Django documentation, 'View' represents how the data is presented to the end-user and not necessarily how it looks. 'Views' in Django reflects the role of traditional controllers in MVC architecture. These classes are located in – '/ar-pedia/views'. In this section, we will discuss how these are implemented.

As discussed in the previous section, requests coming in from the browser are handled by the routing component, which in turn forwards it to the appropriate controller. The controller then handles the logic of the request.

```
class CreatePost(View):
    def get(self, request, pk_scheme):
        argumentation_scheme = SchemeStructure.objects.filter(scheme=pk_scheme).order_by('ordering')
        return render(request, "create_post.html", {"scheme_structure": argumentation_scheme, "scheme": pk_scheme})

    def post(self, request, pk_scheme):
        argumentation_scheme = SchemeStructure.objects.filter(scheme=pk_scheme).order_by('ordering')
        stringBuilder = []
        for tup in argumentation_scheme:
            stringBuilder.append('**')
            stringBuilder.append(tup.section_title)
            stringBuilder.append(':')
```

The snapshot above demonstrates a controller class implementation. Note that it has 2 methods – namely 'get' and 'post', which as its name implies, performs operations based on whether the request routed by the routing component is of type HTTP – Get or Post. In the end, the views sends render request to Django's template engine along with data from the models required for the view and subsequently, an updated web-page is displayed to the end user.

6.2.6 Styling Implementation

Key driving decision behind the user interface of the application involved consistency, scalability (across different devices) and last but not least – ease of use. This section

aims to discuss the key decisions taken – with regards to styling and user experience across the application.

Styling framework

Bootstrap has been known to be a highly efficient and scalable front-end framework in the industry. Its version - 4.3.1 (latest release at time of writing), was used to implement the front-end of the web application. The key driving decisions for doing so are as follows:

- Bootstrap's responsive CSS adjusts to phones, tablets, and desktops, as a result it provides a consistent look and feel across different devices.
- In addition, it is compatible with all modern browsers (Chrome, Firefox, Internet Explorer, Safari, and Opera), including their mobile alternatives.

CSS Styling Unification

CSS styling of web-page components across the entire application is unified in a single CSS file (located in – '/argupedia/static/argupedia/styles.css') and have referenced user interface components in the application via their respective id.

- Therefore, if styling of a specific component has to be changed, for example - buttons, then, only styling code for button attribute in the CSS file has to be changed. Doing so will ultimately reflect across all buttons in the entire application for consistent look and feel. This achieves good maintainability of front-end of the application.

6.2.7 Modelling debates in implementation

Considering debates being made out of attack relation between individual arguments, and also the fact that the grounded extension labelling algorithm is based on directed graph, it was decided that a tree data structure would be best suited to model debates.

A Django based library known as – 'Django MPTT' was used to model underlying tree data structure for debates. Initial argument is regarded as root nodes of the tree, and its child nodes are regarded as attackers of parents and so on and so forth. Another reason 'Django MPTT' was chosen, was because it perfectly integrates with Django models. The code which models this tree relation is shown below:

```
class Entry(MPTTModel):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="author")
    parent = TreeForeignKey("self", null=True, blank=True, on_delete=models.CASCADE, related_name="children")
    content = models.TextField(max_length=4000, default="", help_text="")
    content_formatted = models.TextField(default="")
    title = models.TextField(max_length=100, default="", help_text="", blank=True)
    upvotes = models.ManyToManyField(User, blank=True, related_name="upvotes")
    downvotes = models.ManyToManyField(User, blank=True, related_name="downvotes")
    created_date = models.DateTimeField(default=timezone.now)
    modified_date = models.DateTimeField(blank=True, null=True)
    scheme_used = models.ForeignKey(Scheme, on_delete=models.CASCADE, related_name="scheme_used")
    critical_question = models.ForeignKey(CriticalQuestion, null=True, blank=True, on_delete=models.CASCADE, r
```

Initial argument of debate (regarded as root nodes) has null parents. However, child nodes, attacking the parent node has 'TreeForeignKey' relation as the primary key of the parent node.

Attacks are made on basis of critical questions of parent node. Also, note that the 'CriticalQuestion' model has a Boolean field known as 'Is_Attack_On_Conclusion', which, as its name implies, represents whether the critical question is critiquing the claim of the argument or its premises. Therefore, in a given debate modelled as a tree using 'Django MPTT', we are able to find out which edge between parent and child should be a bidirectional attack based on the basis of the attack, derived by the critical question.

6.3 Implementation of Requirements

In this section, we will discuss how the requirements defined in chapter 3 were implemented and what approach was taken and why.

R1 & R2 - Foundation of web development & front-end

These requirements were implemented as discussed previously in the chapter. To re-iterate - Django framework was used to build the web application coupled with Bootstrap framework for the front-end.

R3 – Secure Login & Registration

In order to implement secure login and registration, Django's built in authentication set-up tool was used. By running the 'superuser create' command in Django console, login and registration was set up. Subsequently, the templates (HTML) of the login and registration pages were redesigned using the layout defined to achieve a consistent user experience.

R4 – Allow structured input to argument by user based on scheme chosen

Model classes were created according to the ERD diagram. Argumentation schemes, as discussed in chapter – 2.1.1, were then inserted in the 'Scheme' model, and their input structure and critical questions were set-up and inserted in the 'SchemeStructure' and 'CriticalQuestion' model respectively. Code snippet defining these models are shown below:


```

class Scheme(models.Model):
    scheme_name = models.CharField(max_length=100)
    def __str__(self):
        return self.scheme_name

class SchemeStructure(models.Model):
    ORDERING = [('1', '1'), ('2', '2'), ('3', '3'), ('4', '4'), ('5', '5'), ('6', '6'), ('7', '7'), ('8', '8'),]
    scheme = models.ForeignKey(Scheme, on_delete=models.CASCADE, related_name="scheme")
    section_title = models.CharField(max_length=100)
    ordering = models.CharField(max_length=50, choices=ORDERING)
    section_description = models.CharField(max_length=100, blank=True)

    def __str__(self):
        return self.section_title

class CriticalQuestion(models.Model):
    related_scheme = models.ForeignKey(Scheme, on_delete=models.CASCADE, related_name="related_scheme")
    is_attack_on_conclusion = models.BooleanField()
    question = models.CharField(max_length=500)

    def __str__(self):
        return self.question

```

When the user attempts to create a debate topic, the scheme names are retrieved from the model and displayed to the user, so that he/she can choose which scheme to start the debate with. Once he chooses that, the structure of that particular scheme is retrieved from the model and displayed as input fields to the user, along with a field to input the title of the topic, coupled with a submit button. Once the user makes his point and submits, its stored in the model as an entry, with parent node as 'null', implying that it is the root of the argument

R5 – Displaying created debate in the application

The home page of the website retrieves nodes from the 'Entry' model, which has no parents (nodes which are starting point in debate) and displays their title as links. When the user clicks on the link, the tree starting at the root node, is retrieved from the models and displaying in a text-based tree form with indentations to highlight which argument in debate is attacking which argument, as shown in the snapshot below:

```

USA Iraq 2003 war
naharul Votes: 1 + - Dec. 1, 2019, 8:18 p.m. counter
In Situation:
Saddam having nuclear weapon
Doing Action:
Invading Iraq
Achieves Goal:
Removal of nuclear weapon
Promotes Value:
World peace

n Votes: 1 + - Dec. 1, 2019, 8:29 p.m. counter delete
Critique Position: The situation described is not true.
Expert Source:
University X war studies department
Expert Area:
Have researched extensively on the USA Iraq war
Expert Assertion:
Saddam does not have nuclear weapon

h Votes: 1 + - Dec. 1, 2019, 8:31 p.m. counter
Critique Position: The expert area of the source is questionable.
Expert Source:
University Y war studies department
Expert Area:
Did research on credibility of University X war studies department
Expert Assertion:
University X war studies department have often been known to have bias

```

R6, R7, R8 & R9 – Critiquing and Countering Arguments

As shown in the snapshot in the last section, users have the option of countering arguments proposed by others. Once the link for countering argument is clicked, critical questions of the scheme used in the node being critiqued is retrieved from the database and displayed in the dropdown. Once a critical question is chosen as a basis, JavaScript is used to display buttons with names containing schemes available. Once the user chooses a scheme for countering an argument, the structure of the scheme is retrieved from the 'SchemeStructure' model and displayed as input fields.

Once the user finishes input of his counter argument according to the scheme, the argument is inserted into the 'Entry' model as a child node of the argument being critiqued, and based on the critical question chosen, we are able to derive if the attack is to be regarded as a bidirectional or unidirectional attack.

Formatting argument point before insertion in model

During insertion of an argument in a model, the argument has to be formatted to be able to make the headers of the scheme chosen, as clear as possible. This is accomplished using a Django based library known as 'Markdown'. When the user submits input structured according to argumentation scheme, the input is converted to a markdown format (<https://en.wikipedia.org/wiki/Markdown>). Subsequently, the format is passed as an input in the Markdown library, which converts it to HTML format, which is stored in the database. The procedure is outlined below:

```
def post(self, request, pk_scheme):
    argumentation_scheme = SchemeStructure.objects.filter(
    stringBuilder = []
    for tup in argumentation_scheme:
        stringBuilder.append('**')
        stringBuilder.append(tup.section_title)
        stringBuilder.append(':')
        stringBuilder.append('**')
        stringBuilder.append('<br>')
        stringBuilder.append(request.POST.get(tup.section_t
        stringBuilder.append('<br>')
    baldur = ''.join(stringBuilder)
    entry = Entry(user=request.user, content=baldur, title=
    entry.save()
```

The code snippet on the left highlights the conversion of user input to markdown format using string builder.

Note that, once it is converted to markdown, the save method is called in the 'Entry' model class, which in turn, calls the 'Markdown' library function to convert markdown format to HTML - highlighting headers of the scheme, along with the data input. This is demonstrated below:

```
def save(self, *args, **kwargs):
    created = True if not self.pk else False
    if not created:
        self.modified_date = timezone.now()
        self.format_content()
    super().save(*args, **kwargs)
    if created:
        self.upvotes.add(self.user)
```

The format content method is called, which, in turn, calls bleach library function to convert markdown to HTML

naharul Votes: 1 + - Dec. 1, 2019, 8:18 p.m. counter

In Situation:

Saddam having nuclear weapon

Doing Action:

Invading Iraq

Achieves Goal:

Removal of nuclear weapon

Promotes Value:

World peace

As a result, in the front-end, we are able to retrieve the formatted content and display in a manner which structures and highlights the scheme headers, as shown on the snapshot on the left.

R10 – User has to be logged in before being able to engage in debate

This was done using built-in Django authentication library, which offers checking of user authentication in Django template language. User authentication is checked and the template is adjusted accordingly.

R11 – Visualization of debate as graph

A JavaScript based graph visualization library known as 'D3' was used to implement the visualization of debates. Most of the logic for the visualization is located in the template file in the location – '/argupedia/templates/visualize.html'.

Initially, the tree starting at the root node (representing start of debate topic) is retrieved in the controller and passed onto the 'visualize.html' template. In the template file, we traverse the tree data passed in from the controller, and initially, format it in JSON. The reason for doing so, is because D3 library takes JSON formatted data as its input.

```
nodes: [
  {% recursetree entries %}
  {
    id: {{node.pk}}, argument: '{{node.content_formatted|safe}}'
  },
  {{ children }}
  {% endrecursetree %}
]
```

This code snippet demonstrates the formatting of the data representing atomic nodes in a debate as JSON.

```
links: [
  {% recursetree entries %}
  {% if node.parent != null %}
  {
    source: {{ node.pk }}, target: {{ node.parent.pk }}, <
  },
  {% if node.critical_question.is_attack_on_conclusion %}
  {
    source: {{ node.parent.pk }}, target: {{ node.pk }}
  },
  {% endif %}
  {% endif %}
  {{ children }}
  {% endrecursetree %}
]
```

This code snippet demonstrates representing edges in the graph in JSON format. Note that in order to represent bidirectional attacks, we check if critical question, is a critique for the claim of the parent. If it is, then we add the bidirectional edge

Once the graph data is formatted, we then call library functions using the data passed in to draw the graph and visualize as a scalable vector graphics (SVG). Full code implementation of the graph visualization can be found in the appendix part of the report. The snapshot below demonstrates that the JSON graph data is being passed in on the function.

```
var simulation = d3.forceSimulation()
    .force("link", d3.forceLink().id(function (d) {return d.id;}).distance(100).strength(0.5))
    .force("charge", d3.forceManyBody().strength(-400))
    .force("x", d3.forceX())
    .force("y", d3.forceY())
    .force("center", d3.forceCenter(width / 2, height / 2));

update(gr.links, gr.nodes, gr.legendVals);
```

R12 – Displaying argument data on mouse-hover over node in graph

Tooltip feature provided by 'D3' library was used to implement this requirement. The data of each argument (formatted to HTML using Markdown), is included in the JSON format for each node and while drawing the nodes - 'mouse hover' event is attached to each of them to display their respective data. The code snippet below demonstrates the implementation of tooltip to display argument information on mouse hover, over node.

```
node = svg.selectAll(".node")
    .data(nodes)
    .enter()
    .append("g")
    .attr("class", "node")
    .on("click", function (d) {window.location.replace("/posts/" + d.id + "/");})
    .on('mouseover', function (d) {tip.html(d.argument);tip.show();})
    .on('mouseout', tip.hide)
    .call(d3.drag()
        .on("start", dragstarted)
        .on("drag", dragged)
    );
```

R13 – Jumping into sub-arguments by clicking on nodes on the graph

This was again implemented by attaching an 'on click' event to individual nodes, so that upon clicking the node, the user will be re-directed to its sub arguments by passing in the 'id' of the node in URL request. The code snippet below demonstrates the attachment of the event to individual nodes when the nodes in the graph is being drawn:

```
node = svg.selectAll(".node")
    .data(nodes)
    .enter()
    .append("g")
    .attr("class", "node")
    .on("click", function (d) {window.location.replace("/posts/" + d.id + "/");})
```

R14 – Marking initial debate argument in debate

This is implemented during formatting of graph data to JSON. When traversing the tree, it is checked whether the node in question – is the root note (is the initial debate

argument). If it is, then the node in the graph is marked with 'S' to illustrate that. This is shown in the snapshot below:

```
node_text: {% if node.parent == null %}"S"{% else %}"{"% endif %}.
```

R15 – Labelling algorithm and marking nodes in graph accordingly

The algorithm for grounded extension labelling, as discussed in section – 2.7.1, was implemented in the 'VisualizeView' controller class. When the user makes URL request to visualize debate, the algorithm is run, and they are labelled into three sets, namely – In, Out, and Undecidable.

These 3 sets are then passed onto the template 'visualize.html', where, as discussed, we iterate over the nodes in the tree to construct a JSON representation which is passed onto 'D3' library for visualization. During these iterations, checks are made for each node to figure out in which of the three sets the argument resides in, and depending on that, they are assigned distinct colours based on their labelling.

The code snippet of the algorithm implemented in the controller is given below:

```
set_in = set() #set in
set_out = set() #set out
set_undec = set() #set undecidable
for entry in root_nodes:
    if (entry.has_children == False and entry.is_root_node() == False):
        if (entry.critical_question.is_attack_on_conclusion == False):
            set_in.add(entry.pk)
while(True):
    prev_set_in = set_in.copy()
    prev_set_out = set_out.copy()
    for entry in root_nodes:
        if entry.pk not in set_in and entry.pk not in set_out:
            okay = True
            for y in entry.get_attackers():
                if y.pk not in set_out:
                    okay = False
                    break
            if okay == True:
                set_in.add(entry.pk)
    for entry in root_nodes:
        if entry.pk not in set_out and entry.pk not in set_in:
            okay = False
            for y in entry.get_attackers():
                if y.pk in set_in:
                    okay = True
                    break
            if okay == True:
                set_out.add(entry.pk)
    if(prev_set_in == set_in and prev_set_out == set_out):
        break
for entry in root_nodes: #append unlabelled sets to set undecidable
    if entry.pk not in set_in and entry.pk not in set_out:
        set_undec.add(entry.pk)
```

Finally, during the formation of JSON representation of graph, each node is assigned distinct colour, depending on which set it resides in (its labelling). The logic of this is shown in the code snippet below:

```
color:{% if node.pk in set_in %}"#CBF6A9"{% elif node.pk in set_out %}"#F6B6A9"{% else %}"#EFF1EE"{% endif %}
```

The full implementation code can be found in the appendix of the report.

R16 – Upvote/Downvote Arguments

Asynchronous JavaScript And XML (AJAX) was used to implement the voting feature for the application. The main reason for doing so, was because it offers immediate feedback on the web-page, without the need to refresh page, which would have been the case if traditional method was used.

‘AJAX was used to construct HTTP-Post request and forward it to the controller. The controller then updates the model and sends back the updated vote count for the argument in question. AJAX then retrieves the updated vote count from the response and amends the appropriate container unit and styling to reflect the updated state. The process described ensures the page is updated instantaneously to reflect the new vote count, without the need to reload/refresh.

R17 – Administration side of the application

As mentioned previously, Django offers automatic instantiation of the administration side of the application. This was initialized by registering the models in the file located at – ‘/argupedia/admin.py’. The code snippet registering the models is shown below:

```
admin.site.register(Entry)
admin.site.register(Scheme)
admin.site.register(SchemeStructure)
admin.site.register(CriticalQuestion)
```

After registration of the models, the ‘create superuser’ command was run on the Django console, to instantiate the administration site of the application. This can be accessed by appending ‘/admin’, at the end of the URL of the website. A snapshot of the admin site is shown below. Through this site, the administrator is able to make changes in the models, as a result is able to amend/add/delete schemes as well as manipulate their structure and critical questions.

Site administration

ARGUPEDIA		
Critical questions	+ Add	Change
Entries	+ Add	Change
Scheme structures	+ Add	Change
Schemes	+ Add	Change
AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change
SITES		
Sites	+ Add	Change

6.4 External Libraries Used and Their Purpose

The external libraries & framework used in the project along with their purpose is listed below:

[Django - 2.2.7](#)

The framework for the project which lays out the foundational structure of the architecture of the application and provides essential security libraries for web application development.

[Django-Mptt - 0.9.1](#)

This library provides a tree data structure which perfectly integrates with Django model. It is used to represent debates in a graph structure in the database

[Bootstrap – 4.3.1](#)

Used to provide a consistent and coherent front-end styling for the application.

[jQuery – 3.3.1](#)

Allows sections in pages of the application to update without the need to reload the entire page. In addition, it is used to make asynchronous requests for upvoting/down-voting an argument.

[Bleach – 3.0.2](#)

This library allows sanitization of HTML content. It allows filtering HTML tags, attributes and styles to a permissible subset defined. It is used when formatting user input of an argument to highlight headers of scheme structure.

[D3.JS – 4.0](#)

This is a JavaScript based library used to visualize debates in the form of directed graph.

[Django-JS-Asset – 1.2.2](#)

Allows insertion of additional attributes when importing scripts in HTML. Is used when importing 'D3'.

[Markdown – 3.0.1](#)

Allows conversion of 'Markdown' formatted text to HTML. It is used when formatting user input of an argument to highlight headers of scheme structure.

[Pytz – 2019.3](#)

Allows accurate and cross platform time zone calculations. It is used to timestamp arguments in debates.

[Six – 1.13.0](#)

Allows bridging the gap between python version 2 and 3. It offers utility functions to smooth the gaps between the Python versions with the purpose of writing Python code consistent with all Python versions.

[SQLParse – 0.3.0](#)

It offers parsing and formatting support for SQL statements.

[WebEncodings – 0.5.1](#)

Allows compatibility with legacy web-content in charset interpretation.

6.5 Instructions on running the project

- Extract the project.
- Navigate to the path of the project using command prompt:
 - `cd <project location>`
- Navigate to the scripts folder using command prompt:
 - `cd Scripts`
- Activate virtual environment:
 - `activate.bat`
- Back out of the folder into the root folder:
 - `cd ..`
- Run the following command:
 - `python manage.py runserver`
- This will start the web-server. A URL will be displayed in the command prompt. Copy the URL into a browser to access the website.

6.5.1 Accessing admin site of the application

In order to access the admin site of the application - append: '/admin', to the home page URL of the website. Doing so, will navigate to the admin login page of the application.

Login credentials for the administration side of the application:

- Username: naharul
- Password: 123

7.0 Testing

This chapter aims to discuss how testing was done throughout the application to ensure it performs as expected

7.1 Testing Approach

Due to the nature of the application being web-based, it was decided traditional unit tests (which tests only atomic functions of the application) would not provide as much value in terms of testing whether use-cases are fulfilled. Although the architecture of the application involves atomic components – such as models, templates, views, however, they are only able to perform use-case operations only when they integrate together.

Use cases such as graph visualization in the application involves complex integration and interaction of components such as routing, views, templates, etc. Thus, unit test is unable to cover these use cases fully. Hence, for the initial release, it was decided that **requirement-based manual testing** approach would be most suitable to test the use cases.

On the other hand, one can argue that individual component interaction testing (such as interaction of routes to controller functions) are very important in development, however, it is worth noting that majority of these functionalities and their integration - are provided by Django framework and are tested by developers of the framework itself. For instance, the integration of routing with views - comes out of the box in the

framework. Therefore, the value of testing these interactions are significantly lower than, suppose if the application was built from scratch without any frameworks.

In addition, one can also argue that there does exist use cases in the application – such as login, registration and CRUD (create, update, read & delete) operations in the administration site of the application, which warrant automated testing, however a large majority of heavy lifting of these use cases are done by the Django framework itself, which is considered to be a robust framework used in the industry, thus well tested. For example, the administration site of the application is automatically generated by Django framework itself and requires only a few lines of code shown below to instantiate it.

```
admin.site.register(Entry)
admin.site.register(Scheme)
admin.site.register(SchemeStructure)
admin.site.register(CriticalQuestion)
```

Another example is the implementation of authentication for login & registration in the application, which was done through the instantiation of 'superuser' through Django console, which automatically set everything with regards to login/registration in the application.

However, it is understood that implementation of automated testing is imperative in the long run to avoid technical debt, however, for the initial release of the project, it was decided that automated tests would not provide much value in terms of testing the use-cases of the application. However, in terms of future work, automated testing is a top priority.

In conclusion, considering the points discussed above, and keeping in mind time constraints, it was decided requirement-based testing would be best suited for the initial release of the application to evaluate if it fulfils the use-cases.

7.2 Requirement Based Testing

Edge cases were prioritized when designing requirement-based tests for use-cases. The test cases designed along with their expected outcome and result are listed below:

Test Description	Expected Outcome	Pass /Fail
Run the website by setting browser setting as mobile and visit each page of website. Scale window up and down to test if it is usable on mobile devices and desktop.	Each element of page is visible to the user and their alignment fits and scales the browser window size on resize.	Pass
Register a new user with invalid email	Show error message that email is invalid	Pass
Register a new user with password length 7	Show error message that password must be of length greater than 8	Pass
Register new user with password as - 12345678	Show error message that password is too common	Pass

Register user with valid email and password, and attempt login.	Login success	Pass
Attempt login with valid registered email but with password which is same length and similar but invalid.	Deny login	Pass
Check registered user password format in database	See password is hashed	Pass
Log out and attempt to upvote/downvote or counter an argument	When the user is not logged in, options for voting and countering argument should not be visible	Pass
Attempt creation of debate topic using all the argumentation schemes in the application.	Text input fields and their headers of arguments are properly structured according to the scheme chosen	Pass
Create debate topic using each scheme and check that contents of the topic are visible to the author, as well as another test user.	Debate topic created is visible in the website and its details can be viewed, including the time at which it was created.	Pass
Log in as a different user and attempt countering the argument using all argumentation schemes available. Test critiquing using all the critical questions available for the original argument.	Ensure critical questions of the original argument are displayed correctly. After selecting critical question, the schemes are displayed, and subsequently, input fields are shown correctly depending on which scheme is chosen. Finally, the counter argument is displayed with proper indentation, with headers highlighted.	Pass
Login as another user and counter the counter argument of the original argument.	Same as the previous test	Pass
Attempt creation of argument with incomplete fields	Do not allow submission without completing the required fields	Pass
Run graph visualization on a range of debates and evaluate whether it is working correctly	Upon visualization, all nodes must reflect the individual arguments in debate and their attack relation must be reflected correctly based on the argument critiqued. In addition, bidirectional attacks must be accurately reflected on the graph based on the critical question basis.	Pass
Test grounded extension algorithm by visualizing a range of debates and going through the algorithm on the graph, evaluating whether it is performing as expected.	Each node on the graph must be labelled and it should be labelled correctly according to the algorithm	Pass
Upvote and downvote arguments and ensure whether vote count in the page is updated instantaneously. Also check database to ensure changes are reflected.	On voting, vote count would increase by one, and on downvoting, it will decrease by 1. On clicking an already upvoted/downvoted argument, the upvote/downvote will be	Pass

	cancelled and the vote count will be removed, and the UI will reflect the change. In addition, change is reflected on database for each operation	
--	---	--

8.0 Evaluation

As the project is meant to be used by users around the globe, the focus on evaluation was mainly on usability to understand whether people feel comfortable using the application and using argumentation schemes to debate.

As a result, the plan of evaluation involved – Firstly, interviewing people and in a controlled setting, asking them to perform various tasks in the application and noting the time taken and number of times they were stuck - in order to gather quantitative usability data. Secondly, the plan was to interview a different set of people (in same controlled lab environment) and instructing them to ‘think-aloud’ while performing tasks, to gather quantitative data for evaluation. Thirdly, the plan was to go through the two data set and analyse to find in what aspects of the system had room for improvement.

In addition, the plan also involved deploying the application among a small group of people (as a technology probe) and instructing them to debate using the application over a specific number of days, and in the end, ask them for their feedback on what can be improved, and last but not least, use a System Usability Scale [24] to evaluate their view on the application.

However, due to the ‘Covid-19’ pandemic, it became increasingly difficult to evaluate the application based on the plan outlined above. As a result, a decision was made to evaluate usability of the application based on Nielsen’s ten usability heuristics [25].

8.1 Evaluation based on Nielsen’s ten usability Heuristic

Usability heuristics are pointers for finding usability problems in the system. It involves reviewing the user interface of the application to evaluate and judge its adherence to accepted standards of usability.

8.1.1 Visibility of System Status

This heuristic tells us that the system should always keep the user informed on what is going on, through appropriate feedback within reasonable amount of time. According to Nielsen, it helps the user build a relationship of trust with the system.

During evaluation, it was noted that when debates get quite large, visualizing them as a graph takes some time, however, the visualization page is simply blank while the graph loads, and provides no feedback regarding its state to the end-user. Hence, it violates the heuristic.

To improve on this, it is noted that implementing a visual feature, which indicates the progress of loading as the graph instantiates will make this element of the application more in alignment with the heuristic.

8.1.2 Match between system and the real world

This heuristic states that the system should speak the user's language, as in, use words, phrases and concepts familiar to the user, as opposed to system-oriented terms.

When evaluating this heuristic against the system, it was acknowledged that the application has numerous rooms for improvement. For instance, the application uses technical jargons such as 'argumentation schemes', 'critical questions', which may not be familiar to the end user.

To improve on this, it is noted that a need to convert technical jargons to relatable concepts will be beneficial to the end-user. For example, the technical jargon 'critical question', could be changed to 'critique basis', to make it more relatable for the user.

It is acknowledged that after addressing these issues, a thorough usability test involving real potential end-user is needed to judge effectiveness of the improvement. If further issues are identified during usability tests, these will have to be incorporated on an iterative basis.

8.1.3 User control and freedom

The heuristic states that users will inevitably choose to perform operations by mistake and require a clearly marked exit to back out of the operation.

When evaluating this heuristic against the system, numerous violations were identified. There could be scenarios where users input a counter argument to an argument and after submission, identifies errors in his point, and would like to make changes.

One solution could be a simple edit feature of arguments; however, this opens doors of manipulating debates. For instance, a user could make a point, however, an attacker of that argument makes a better point, which could entice the user to edit his argument to get an unfair advantage.

Therefore, a reasonable solution could be to give the user a short time-frame until his argument is published/goes live. During the time frame, the user could make changes if he thinks he has made some errors in his submission. The question now remains - what timeframe to choose? To accurately judge this, it would be better to gather the viewpoint of potential users of the system.

In addition, when the user chooses too to visualize a debate, a button to exit out of the visualization does not exist in the application. Furthermore, during counter argument, once the user chooses a scheme to base his point on – there is no option or back button - to choose a different scheme, in case the user chose a scheme by mistake. Although, most browsers accompany a 'back' button, however, the application itself lacks in this regard.

8.1.4 Consistency and standards

This heuristic states that the application should follow consistency in different uses of visual elements. In addition, it also makes it clear that locations of visual elements of the application, should follow user's expectation.

During evaluation of this heuristic against the application, it was concluded that the application generally is in alignment with this heuristic. Button styles and their animations are consistent across all instances of buttons in the application. In addition,

navigation locations, such as pagination (usually expected to be at bottom of a list) and navigation bar (usually expected at top of website) is consistent across the application.

8.1.5 Error prevention

This heuristic state that measures should be taken to prevent the user from committing errors during use of the application

During evaluation of the application against this heuristic, it was noted that some aspects are in alignment, whereas others fall behind. For instance, form validations are implemented well across the application to guide the user from submitting erroneous data, which aligns with this heuristic, however, there is room for improvement. For example, there could be instances where the user accidentally clicks 'submit', to submit an argument without completion. It could be improved by implementing user confirmation, especially at end decision points of the application.

8.1.6 Recognition rather than recall

This heuristic state that the user interface of the application should be designed in a way such that users are able to apply what they already know to hit the ground running and get familiar with the application fast, during usage.

During evaluation of the application against this heuristic, it was concluded that the application is generally in alignment with this heuristic. For instance, navigation locations, such as pagination (usually expected to be at bottom of a list) and navigation bar (usually expected at top of website) is consistent across the application. Another example is the existence of button for 'log out' in the dropdown of the name of the person logged in. Furthermore, clickable links in the application, is highlighted in blue, - just as regular links in other websites

8.1.7 Flexibility and efficiency of use

This heuristic state that 'accelerators' such as keyboard shortcuts should be used and made apparent within the application to cater for users who are expert users of the system.

During evaluation of the application against this heuristic, it was concluded that to cater for potential regular users of the system, keyboard shortcuts for configuring magnifying extent (zoom) of the graph visualization would be a good feature to have for the application. This would enable regular users of the system to effectively magnify the visuals of the graph to his use.

8.1.8 Aesthetic and minimalist design

This heuristic state that users should not be presented unnecessary information. Its better if textual information can be minimized through the use of easy to understand visual elements, without compromising on information loss.

Upon evaluation of the application against this heuristic, it was concluded that the application generally follows the criteria laid out. For instance, in graph visualization, the labelling of the nodes is denoted by a simple legend on the top right, thus minimizing excess information. In addition, when browsing through a debate, attack relations of arguments are visually implied by indentation (similar to the style on websites such as reddit), thus adhering to minimalist design.

8.1.9 Help users recognize, diagnose, and recover from errors

This heuristic state that, in case an user encounters an error in the system, he/she should be shown error messages which are productive and understandable to non-technical person.

The application is generally in alignment with this heuristic. Error messages for form validations are easily understandable to the end-user. In addition, Django not only automatically handles errors with regards to server, but also, displays error pages which minimize technical jargon to the end-user.

8.1.10 Help and documentation

This heuristic state that applications should always provide a help/documentation section for the end-user.

The application generally lacks help aimed at the end-user. For example, a new user may not understand the scheme structure of a particular argumentation scheme. A solution is to implement a help icon beside each argumentation scheme, which if the user hovers mouse over, displays a tooltip explaining the structure of the scheme with examples so the user feels comfortable using the scheme. In addition, during graph visualization, the legend denotes labelling of nodes in terms of - 'In', 'Out' and 'Undecidable'. Having a mouse hover tool tip explaining what the labelling mean would make it easier for an unfamiliar user to understand the system.

8.1.11 Conclusion and Reflection

This heuristic state that applications should always provide a help/documentation section for the end-user.

The application generally lacks help aimed at the end-user. For example, a new user may not understand the scheme structure of a particular argumentation scheme. A solution is to implement a help icon beside each argumentation scheme, which if the user hovers mouse over, displays a tooltip explaining the structure of the scheme with examples so the user feels comfortable using the scheme. In addition, during graph visualization, the legend denotes labelling of nodes in terms of - 'In', 'Out' and 'Undecidable'. Having a mouse hover tool tip explaining what the labelling mean would make it easier for an unfamiliar user to understand the system.

8.2 Evaluation of Argupedia Debates

While the application does enforce structured debates by means of argumentation schemes, it however still allows scope of people making false claims. For example, a person with malicious intent might use the expert opinion scheme to falsely make up an expert to support his false argument. The application does not enforce checking facts.

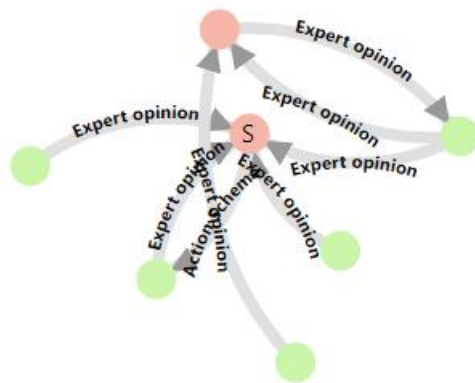
A potential solution for this issue could be through incentivising moderators in the platform, who independently fact-checks argument and raises potential concerns. However, it is acknowledged that manual moderation is not scalable in the platform, as debates could potential grow significantly. Therefore, Further exploration regarding automation of fact-checking is needed.

It is worth noting that the evaluating accepting arguments through Dung's abstract argumentation framework is considered as non-monotonic. This means that an

argument could be regarded as accepted simply because no other argument has challenged it yet. Thus, to get the most out of the labelling algorithm, participants have to regularly challenge and actively debate on the topic. However, if participants are not actively debating, then a potentially illogical argument could be regarded as 'accepted', simply because no one has challenged it yet. This could potentially lead to people, who do not know much about a topic, regarding a weak argument as accepted.

8.3 Evaluation of Visualization of Debates

During evaluation, it was noted that, if debates get fairly long, the initial arrangement of nodes in the directed graph makes it difficult to get an overview. An example debate visualization shown on the snapshot on the left demonstrates initial arrangement of nodes. Although 'D3', allows users to rearrange position of nodes by dragging nodes around, however, it would be significantly better if the position of the nodes were automatically arranged to their optimum position to make the graph easier to understand at a glance, instead of having to manually adjust the position.



8.4 Evaluation of Project

The project's background research conducted helped to clarify the challenges and potential ways to address them. Design of the project helped visualize and map out the foundation of the implementation, however, emphasis on user experience design was lacking. It could be addressed in the future through the development of prototype designs and evaluating them on a set of potential users to understand which design would cater best towards user experience.

Subsequently, Iterative development methodology employed for the project provided versatility to fulfil the requirements and accommodate new ones. However, it is acknowledged that for a project such as this, which is likely to significantly grow in the future, the lack of automated testing was a significant shortcoming. Although thorough manual testing was conducted with edge cases in mind, however, automated testing is a necessity to avoid the build-up of technical debt in future development iterations.

In conclusion, although requirements initially set out for the project were fulfilled through research and implementation coupled with testing, however the application falls short in terms of user experience in some aspect which could be improved upon.

9.0 Legal, Social, Ethical and Professional Issues

The Code of Ethics & Code of Good Practice issued by the British Computing Society (BCS) was of considerable interest during the design and execution of the project. It was to ensure the solution being built is in adherence to professional standards, disregarding which - could open doors for potential significant legal and ethical consequences.

According to professional standards, the usage of content without owner's permission is deemed immoral. A number of open-source libraries/frameworks have been used in the project to speed up development and produce higher quality work- as made clear in chapter - 6.4. The project has taken careful consideration to ensure that external libraries used throughout - are explicitly stated and credit is given where due.

BCS code of conduct states that utmost care is to be ensured for privacy and security of user's data. The usage of Django's built in security and authentication system has ensured that authentication and user data protection has been built according to recent standards in the industry.

In addition, the code of conduct states that software services should try to facilitate the incorporation of all sectors into society whenever possible. Keeping this into mind, the application was designed to be useable and scalable in all types of browsers on different devices, including mobile. This is to ensure a wider reach and prevent discrimination and make the application accessible to everyone.

Furthermore, it has been acknowledged in chapter 8.3 that the initial release of the application does not implement any fact-checking measures for arguments. However, the strives to be transparent about this issue, by explicitly letting the user know that the arguments are not fact-checked automatically and urging them to conduct research independently. Numerous potential solutions have been devised, including the use of moderators in the application, which will be addressed in future work of the application.

General Data Protection Law (GDPR) states that data considered as personal to users of the system should not be collected unless absolutely necessary. In addition, if the service absolutely requires personal data collection, then users should be adequately informed about its use. In addition, the system should only hold data, up until to the point it is necessary. Taking this regulation into account, the application simply requires a user's chosen username and password to set up an account and engage in debates. In addition, Django's security library automatically ensures password is hashed in the database. Therefore, no personal data is collected in the application.

10.0 Conclusion & Future Work

As discussed in chapter – 8, the initial release of Argupedia is not perfect and has numerous rooms for improvement. Referring to the shortcomings of the application, and also looking into the future, this section aims to discuss possible future work on the application which could provide value to the end-user.

As discussed in chapter 6.2, significant effort was made to build the application in a way to optimize for greater maintainability and extendibility. Considering this advantage, the following key improvements and future work on the project was identified:

- At the moment, the application only facilitates debates where people criticize and counter each other's argument, however, future work could include a feature which enables people to support arguments of fellow participants as well. However, further research is needed to explore ways of tweaking the abstract argumentation framework to accommodate supporting relation.
- For arguments labelled as 'undecidable' by the labelling algorithm, a feature could be implemented which decides the labelling based on vote count for each two undecidable argument. However, the application must be transparent, as in - arguments decided by vote count must be clearly marked to make it clear to the end-user that the labelling for the argument has indeed been decided by vote count.
- It is clear there exists scope for improving usability of the application. As a result, moving forward – Usability evaluation study could be conducted with a group of participants regarded as potential user of the system. The data gathered from the study could be synthesized and conclusions inferred to improve user experience design of the system.
- Further research to minimize the scope of providing false claims in debate is needed. At the moment, the application fails to provide a fact-checking mechanism to verify premises of an argument (for instance, verification of expert source in an expert opinion scheme backed argument). A moderation system could be implemented which allows moderators of the application to fact-check and potentially mark such arguments to let the users know that it requires verification. It is acknowledged that manual moderation is not scalable, hence further research is needed to look into ways through which we could potentially flag such arguments to make the job easier.
- A feature for archiving debates could be implemented. Upon archive, a function for downloading transcript of debate would prove useful to have as a record which could be analysed later to gain more insight regarding an issue.

11.0 Bibliography

- [1] Statista (2019, November 20). Reddit - Statistics & Facts [Online]. Available: <https://www.statista.com/topics/5672/reddit/>
- [2] Ginny Marvin (2018, September 17). Quora introduces Broad Targeting, says audience hits 300 mil-lion monthly users [Online]. Available: <https://marketing-land.com/quora-introduces-broad-targeting-says-audience-hits-300-million-monthly-users-248261>
- [3] Amazon (2019, December). The top 500 sites on the web [Online]. Available: <https://www.alexa.com/topsites>
- [4] Atkinson K, Bench-Capon T, McBurney P. PARMENIDES: facilitating deliberation in democracies. *Artificial Intelligence and Law*. 2006 Dec 1;14(4):261-75.
- [5] SubReddit Stats (2019, December 4). Changemyview Stats [Online]. Available: <https://subredditstats.com/r/changemyview>
- [6] Karin Osvaldsson, "Bullying in Context: Stories of Bullying on an Internet Discussion Board," Dept. of Child Studies, Linköping University, 2011
- [7] Plous Scott. *The psychology of judgment and decision making*. McGraw-Hill Book Company; 1993.
- [8] Van Eemeren F, Van Eemeren FH, Grootendorst R. *A systematic theory of argumentation: The prag-ma-dialectical approach*. Cambridge University Press; 2004.
- [9] Dung PM. On the Acceptability of Arguments and its Fundamental Role in Non-monotonic Reasoning and Logic Programming. *InIJCAI* 1993 Aug 28 (Vol. 93, pp. 852-857).
- [10] Gerhard Brewka, Class Lecture, Topic: " Handling Exceptions in Knowledge Representation: A Brief Introduction to Nonmonotonic Reasoning", Computer Science, University of Leipzig, Leipzig, May 2017
- [11] Caminada M. *A gentle introduction to argumentation semantics*. Lecture material, Summer. 2008.
- [12] Baroni, P., Caminada, M. and Giacomin, M., 2011. An introduction to argumentation semantics. *The knowledge engineering review*, 26(4), pp.365-410.
- [13] Peter McBurney, Class Lecture, Topic: "Lecture 7: Argumentation I", Computer Science, King's College London, London, November 2019.
- [14] P. M. Dung, "On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming," Division of Computer Science, Asian Institute of Technology, 1995.
- [15] Modgil S, Caminada M. Proof theories and algorithms for abstract argumentation frameworks. In *Argumentation in artificial intelligence 2009* (pp. 105-129). Springer, Boston, MA.

- [16] Dagan I, Dolan B, Magnini B, Roth D. Recognizing textual entailment: Rational, evaluation and approaches—erratum. *Natural Language Engineering*. 2010 Jan;16(1):105-.
- [17] Cabrio E, Villata S. Combining textual entailment and argumentation theory for supporting online debates interactions. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* 2012 Jul (pp. 208-212).
- [18] Walton D., “Chapter 3: Argumentation Schemes” in *Fundamentals of critical argumentation*. Cam-bridge University Press; 2005 Oct 31.
- [19] Walton D, Reed C. Argumentation schemes and defeasible inferences. In *Workshop on computation-al models of natural argument, 15th European conference on artificial intelligence* 2002 Jul 22 (pp. 11-20).
- [20] Live Science (2017, July 25). Deductive Reasoning vs. Inductive Reasoning [Online]. Available: <https://www.livescience.com/21569-deduction-vs-induction.html>.
- [21] Atkinson, K., Bench-Capon, T. and McBurney, P., 2006. PARMENIDES: facilitating deliberation in democracies. *Artificial Intelligence and Law*, 14(4), pp.261-275.
- [22] Atkinson KM. What should we do? Computational representation of persuasive argument in practical reasoning.
- [23] Django FAQ General (2020, April 01). Available: <https://docs.djangoproject.com/en/3.0/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names>
- [24] Nathan Thomas, (2015, July). “How to Use the System Usability Scale (SUS) To Evaluate the Usability of Your Website”. Available: <https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/>
- [25] Jacob Nielsen, "Chapter 2, Heuristic Evaluation" in *Usability Inspection Methods*, Jacob Nielsen and Robert L. Mack, John Wiley & Sons, New York, 1994