



# **Argupedia: a Platform for Easy, Structured Debates Online**

Final Project Report

Author: Jonathan Damico

Supervisor: Dr Sanjay Modgil

Student ID: 1911456

April 8, 2022

## **Abstract**

Argupedia is a platform built from the ground up to facilitate online arguments and debates. Argupedia is hardly the first platform to attempt this but Argupedia differentiates itself by utilising an argumentation system to keep debates structured. The argumentation system used forces users to choose a schema and work within that schema in order to generate their arguments, keeping users on topic and maintaining a clean debate.

Argupedia is built with a web-first framework and implements several software engineering practices to ensure that its codebase is secure, well-written, working, and scalable.

### **Originality Avowal**

I verify that I am the sole author of this report, except where explicitly stated to the contrary.  
I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Jonathan Damico

April 8, 2022

### **Acknowledgements**

I would like to thank my supervisor Dr Sanjay Modgil for his support throughout the project. His guidance was invaluable during all stages of Argupedia's development.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Existing Applications . . . . .	4
2.2	Abstract Argumentation System . . . . .	5
2.3	Labelling Arguments . . . . .	7
2.4	Supporting Arguments . . . . .	7
2.5	Argument Schemes and Critical Questions . . . . .	7
<b>3</b>	<b>Requirements</b>	<b>10</b>
3.1	User Requirements . . . . .	10
3.2	System Requirements . . . . .	12
<b>4</b>	<b>Design and Implementation</b>	<b>13</b>
4.1	System Architecture . . . . .	13
4.2	Backend . . . . .	13
4.3	Frontend . . . . .	20
4.4	Software Engineering Practices . . . . .	26
<b>5</b>	<b>Legal, Social, Ethical and Professional Issues</b>	<b>30</b>
5.1	Legal Issues . . . . .	30
5.2	Social Issues . . . . .	31
5.3	British Computing Society Code of Conduct . . . . .	31
<b>6</b>	<b>Results/Evaluation</b>	<b>32</b>
6.1	Quality Evaluation . . . . .	32
6.2	Software Testing . . . . .	34
<b>7</b>	<b>Conclusion and Future Work</b>	<b>38</b>
7.1	Future Work . . . . .	38
	Bibliography . . . . .	40
<b>A</b>	<b>Argupedia API Documentation</b>	<b>41</b>
A.1	Endpoints . . . . .	41
<b>B</b>	<b>Argument Schemas</b>	<b>53</b>
B.1	Argument Types used by Argupedia . . . . .	53

<b>C Argument Critical Questions</b>	<b>57</b>
C.1 List of Critical Questions By Argument . . . . .	57
<b>D User Guide</b>	<b>61</b>
D.1 Installation & Setup . . . . .	61
D.2 Application Use . . . . .	64
<b>E Source Code Listings</b>	<b>78</b>
E.1 Argupedia API Source Code . . . . .	85
E.2 Argupedia App Source Code . . . . .	368

# Chapter 1

## Introduction

The internet, perhaps unsurprisingly, has increased the velocity with which information can be shared and propagated around the world and has opened the floodgates for open public discourse. Websites such as Reddit, Twitter, and Quora have emerged to capture the market for this discourse but have typically done a poor job of structuring the conversation in a way conducive to a productive debate.

Formal argumentation schemes have been created and have existed for a long time and we have found ways to implement these argumentation schemes into real-world formalities which advance our knowledge and understanding of the world. Competitive debating, parliamentary debating, and debating societies are just a few examples of arenas where formal argumentation schemes are applied toward relevant societal issues and many examples of these arenas have existed and refined themselves for several hundred years. Similarly, these frameworks have been applied in academia where many research papers and argumentative essays must conform to an argumentation framework to demonstrate, prove, or disprove findings. These written arguments have similarly existed and refined themselves over a long period.

The internet, however, is still devoid of a platform or service which similarly embraces argumentation and existing platforms allow for their users to argue in a completely unstructured way. Argupedia seeks to be an online platform that would formally facilitate discussion and debate, using an argumentation framework, not too dissimilar from one of the aforementioned debating arenas.

# **Chapter 2**

## **Background**

This section of the report will focus on the existing platforms which allow for argumentation and then will focus on the relevant literature on argumentation which is later implemented in the Design & Implementation section of the report.

### **2.1 Existing Applications**

There are a wide variety of applications that allow a user to engage in debates and arguments online. These platforms are extremely diverse both in terms of their popularity but also in terms of how structured the arguments on their platforms are.

#### **2.1.1 Platforms for Unstructured Argumentation**

Many websites on the internet are extremely popular and can facilitate large arguments such as Twitter, Reddit, and Quora. These websites have, intentionally or unintentionally, become de-facto "town squares" and constantly facilitate discussions and arguments. These platforms do not do a good job of structuring arguments or encouraging the user to structure their thoughts. This is largely because these platforms started as discussion forums or social media platforms and, due to their incredible popularity, have become centres for argumentation over time. Their popularity makes them worth mentioning as any product in the space of facilitating online arguments must compete with these platforms.

### 2.1.2 Platforms for Structured Argumentation

Additionally, several platforms are focused on providing a platform designed for argumentation. The most popular of these platforms is Kialo which allows users to engage in arguments in a structured way. Kialo makes use of user feedback to decide which arguments are winning and implements no formal argumentation system, requiring the user to input their arguments ad-hoc.

To keep arguments and discussions organized and consistent on Kialo, their platform employs moderators which ensure that any new argument added to a discussion is coherent.

### 2.1.3 Differentiation

Argupedia attempts to use a formal argumentation system to structure argumentation. By using such a system, Argupedia will remove the "popularity bias" which can occur on a site such as Kialo where popular arguments may win even if their attackers are technically correct. Using an argumentation system also allows more complicated analysis to be run on collections of arguments and removes the need for moderators to ensure that arguments have been constructed correctly. Lastly, employing an argumentation system. will keep any arguments or debates on Argupedia focused as users will be guided through the process of creating acute arguments that attack specific elements of existing arguments.

## 2.2 Abstract Argumentation System

This project relies heavily upon the Dung argumentation framework and its extensions. Dung developed his framework based upon how humans argue with each other in the real world: whichever party has the final word in an argument wins.

A Dung argumentation framework is defined as a pair

$$AF = \langle AR, attacks \rangle \tag{2.1}$$

where AR is a set of arguments and attacks is a binary relation on AR, i.e.  $attacks \subseteq AR \times AR$ .  
[2]

In this definition, for two arguments  $A$  and  $B$ ,  $attacks(A, B)$  represents that  $A$  is an attack against  $B$ .

This framework can therefore be represented as a directed graph with  $AR$  representing all

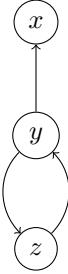


Figure 2.1: Example argumentation framework

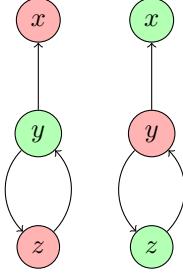


Figure 2.2: All preferred extensions of figure 2.1

nodes within the graph and *attacks* representing the set of edges for the

From this basic definition, the acceptability of arguments can be determined. Logically, an argument is acceptable if it can be defended against all of its attackers. This leads to the definition of an acceptable argument:

An argument  $A \in AR$  is *acceptable* with respect to a set of  $S$  arguments iff for each argument  $B \in AR$ , if  $B$  attacks  $A$ ,  $B$  is attacked by some argument  $C \in AR$ . [2]

An entire set of arguments is *admissible* iff each argument in  $S$  is acceptable with respect to  $S$ . [2]

A preferred extension of an argumentation framework is a maximal admissible set of an argumentation framework. As an example, take the argumentation framework in figure 2.1 and its preferred extensions represented in figure 2.2. In figure 2.2, all green nodes represent arguments that have been labelled as IN and all red nodes represent arguments that have been labelled as OUT. In both cases, all arguments are acceptable and therefore, both labellings are preferred.

A grounded extension of an argumentation framework contains all arguments which are guaranteed to be accepted.

## 2.3 Labelling Arguments

Arguments in the grounded extension can be labelled as IN, OUT, or UNDEC. Arguments can be labelled according to the following conditions:

- $x$  is legally IN iff  $x$  is labelled IN and every  $y$  that attacks  $x$  is labelled OUT
- $x$  is legally OUT iff  $x$  is labelled OUT and there is at least one  $y$  that attacks  $x$  and  $y$  is labelled IN
- $x$  is legally UNDEC iff  $x$  is labelled UNDEC and not every  $y$  that attacks  $x$  is labelled OUT, and there is no  $y$  that attacks  $x$  such that  $y$  is labelled IN

These rules encapsulate the basic rules of argumentation: an argument can only be IN if all of its attackers are OUT and an argument that is OUT must be attacked by at least one argument which is IN.[4]

## 2.4 Supporting Arguments

To fully represent how arguments develop, supporting arguments must also be accounted for. With the Dung argumentation framework and the supported labelling algorithm, arguments and their supporting arguments should be grouped. An attack on a supporting argument can also be considered to be an attack on the argument it supports.

Supporting arguments can be considered to be extensions of such an argument. Consider two arguments,

Argument A:  $[(s, s \rightarrow p), p]$  with premise  $p$  and assertion  $q$ .

Argument B:  $[(p, p \rightarrow q), q]$  with premise  $p$  and assertion  $q$ .

where argument A is supporting argument B.

This can be represented together as one large argument,  $[(s, s \rightarrow p, p \rightarrow q), q]$ .

With this reduction, an attack on one argument can be considered to be an attack on all.

## 2.5 Argument Schemes and Critical Questions

The final extension of the argumentation framework is the several argument schemes and critical questions which formalize common arguments that an individual might make.

The argumentation schemes and critical questions used for this project are based on the work of Atkinson[3] and Philosophical Disquisitions[1] which themselves are based on an extremely thorough investigation of argumentation by Walton[5].

Each argumentation scheme contains a set of propositions that make up the argument. For example, the appeal to expert opinion argument is outlined as follows:

Source  $E$  is an expert in domain  $D$  containing proposition  $A$   $E$  asserts that  $A$  in domain  $D$  is boolean  $B$ .[1]

By filling in a value for  $E$ ,  $D$ ,  $A$ , and  $B$  a full argument can be generated:

$E$  = The director of the NHS

$D$  = Medicine

$A$  = There is a pandemic

$B$  = True

The director of the NHS is an expert in medicine and asserts that it is true that there is a pandemic.

And finally, with these values, critical questions can be generated for the given argumentation schema. For example, the critical questions for the appeal to expert opinion argument scheme are:

- How credible is  $E$  as an expert?
- Is  $E$  an expert in the field that  $A$  is in?
- What did  $E$  assert that implies  $A$ ?
- Is  $E$  personally reliable and trustworthy? Do we have any reason to think  $E$  is less than honest?
- Is  $A$  consistent with what other experts assert?
- Is  $E$ 's evidence based on evidence?

Filling in a value for  $E$ ,  $D$ ,  $A$ , and  $B$  can generate a full set of critical questions for this argument.

- How credible is the director of the NHS as an expert?
- Is the director of the NHS an expert in the field that there is a pandemic is in?
- What did the director of the NHS assert that implies there is a pandemic?

- Is the director of the NHS personally reliable and trustworthy? Do we have any reason to think the director of the NHS is less than honest?
- Is there is a pandemic consistent with what other experts assert?
- Is the director of the NHS's evidence based on evidence?

The full set of argumentation schemes and critical questions used by Argupedia are described in Appendices B and C respectively.

# **Chapter 3**

# **Requirements**

The platform is designed to be used by multiple users who wish to have structured arguments with each other. As such, there are two sets of requirements for this project: user requirements and system requirements. The user requirements will outline what high-level activities and functions the user should be able to execute on the platform and the system requirements outline the capabilities the system needs to support the user requirements.

## **3.1 User Requirements**

1. Users must be able to explore arguments
  - (a) Users must be able to see a graphical representation of the argument framework the argument belongs to.
  - (b) The user must be able to jump around the argument framework by selecting arguments within that representation. Selecting an argument should reveal the details of that argument.
  - (c) The graphical representation of the argument framework should visually indicate which arguments are in, which arguments are out, and which arguments are undefined. The representation should also visually indicate whether an argument is supporting or attacking another argument within the framework.
2. Users must be able to create new arguments.
  - (a) The user must be able to select an argument scheme to use and should be able to fill in the elements of that argument scheme.

- (b) The user must see a rough text representation of their argument based on what they have entered. The user should be able to modify this representation as an automatically generated representation may not always be grammatically or syntactically correct.
- (c) If the user is creating an attacking or supporting argument, they must be presented with a set of critical questions which they can use to inspire their new argument.
- (d) Users should be able to edit arguments after they have been created if they are the author of that argument.

3. Users must be able to create comments on arguments

- (a) Users should be able to add new comments to arguments.
- (b) Argument comments should be ordered by when they were created.
- (c) Users should be able to edit or delete comments that they have authored.

4. Users must be able to see content that they have created on the platform.

- (a) Users should be able to navigate to a page that will allow them to see all content they have created.

5. Users must be able to delete content that they have created on the platform.

- (a) When the user is viewing content that they have authored, they should have the option to delete this content.
- (b) When a user account is deleted, all content that they have authored on the platform should also be deleted. The notable exception to this requirement is that arguments should not be deleted as this would disrupt the state of existing arguments on the platform haphazardly. In this case, the arguments which are authored by the user should no longer display an author or should say that the author no longer exists.

6. Users must be able to edit their profile details

7. (a) Users must be able to delete their accounts.  
(b) Users must be able to update parts of their profile, such as their name or their biography.

8. Users must be able to view a basic profile of other users

- (a) Users must not be able to see sensitive information about other users such as their email.

## **3.2 System Requirements**

1. Sensitive user data and login credentials must be stored in an encrypted format.
2. The platform must be able to authenticate users to prevent users from modifying objects that they do not have the right to modify.
3. The platform must be able to analyse argument frameworks as they are created and modified.
  - (a) The backend should be able to add new arguments to existing argument frameworks.
  - (b) The backend must be able to label argument frameworks automatically and must re-label argument frameworks whenever they are augmented.
4. The platform must comply with relevant data protection laws.

# Chapter 4

# Design and Implementation

This section will walk through the aspects of Argupedia's design and how that design was implemented. Two distinct programs were designed for argupedia: a standalone frontend and a standalone backend. These two products are designed to fit well with each other but are hosted and managed independently of each other. This section will explore each product and will also investigate the software development practices used while making Argupedia.

## 4.1 System Architecture

As demonstrated in figure 4.1, the system makes use of AWS services. Additionally, the project is split starkly between the frontend and the backend. This chapter of the report will focus on the frontend and the backend separately.

## 4.2 Backend

The backend is written using Node.JS using TypeScript to ensure that objects are statically typed.

There are three main layers to the backend:

- API Layer
- Helper Layer
- Database Interface Layer

The API layer contains all handlers for all exposed endpoints. This layer is responsible for

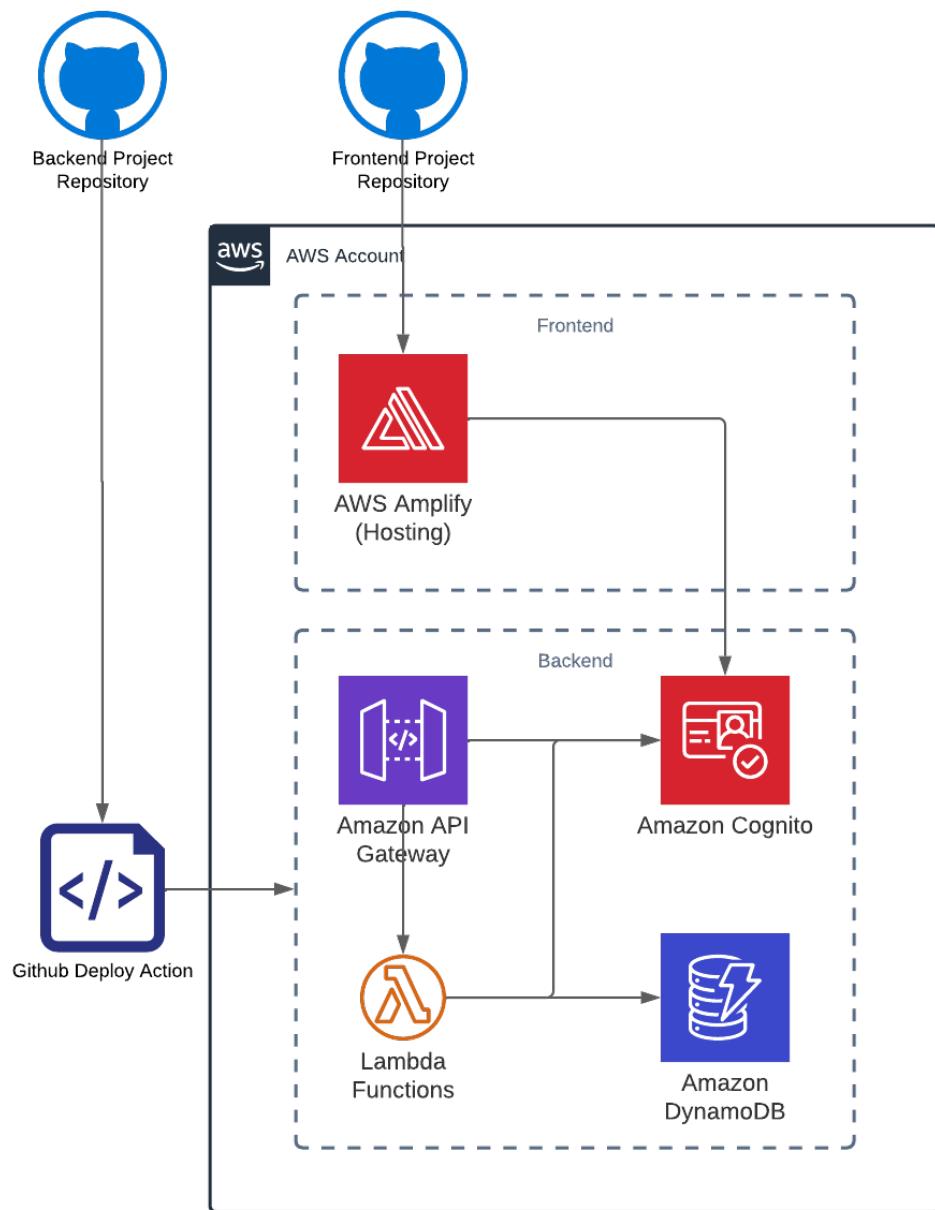


Figure 4.1: Overall System Architecture Diagram

authenticating the user if the endpoint requires authorisation, validating the parameters of the request, and calling the appropriate methods in the helper and database layers.

The helper layer contains procedures and methods which are shared between handlers or which are particularly complicated. This layer, therefore, ensures that there is low code duplication in the codebase. One example of a method in the helper layer is the labelArguments method which is responsible for labelling an argument framework after it has been augmented.

Lastly, the database interface layer contains methods that directly interact with the database. This layer validates data that is entering the database, ensures that it is structured per the defined database schema, and processes any retrieved data from the database.

#### 4.2.1 Database

The database has three main tables: one to store user details, one to store information about arguments, and one to store comments. The relationships between the databases are shown in the entity relation diagram shown in figure 4.2

```
type UserTableEntry = {
  userId: string;
  authoredArgumentIds: Array<string>;
  authoredCommentIds: Array<string>;
  votedArgumentIds: Array<string>;
  displayName: string;
  biography: string;
  profilePictureURL: string;
  lastSignedOn: string;
};

type ArgumentTableEntry = {
  argumentId: string;
  argumentType: string;
  argument: Argument;
  argumentText?: string;
  authorId: string;
  createdDate: string;
  modifiedDate: string;
  supportsArgumentId?: string;
  supportedByArgumentIds: Array<string>;
  attacksArgumentId?: string;
  attackedByArgumentIds: Array<string>;
  upvotedByUserIds: Array<string>;
  downvotedByUserIds: Array<string>;
  label: 'IN' | 'OUT' | 'UNDECIDED';
  commentIds: Array<string>;
  argumentDepth: number;
};
```

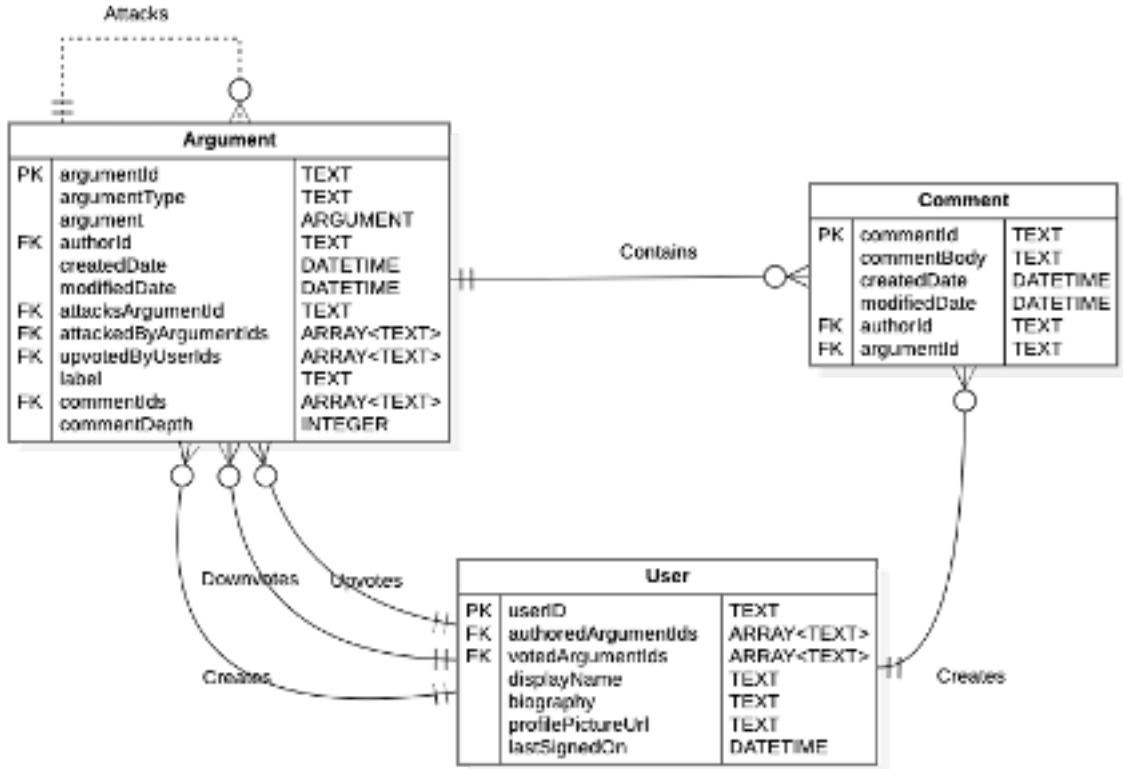


Figure 4.2: Database Entity Relation Diagram

```
type CommentTableEntry = {
  commentId: string;
  commentBody: string;
  createdDate: string;
  modifiedDate: string;
  authorId: string;
  argumentId: string;
};
```

This project uses DynamoDB (AWS's NoSQL database) for its database. This was chosen primarily because the database needs to handle multiple types of argument schemes and working in a NoSQL database will allow these to be stored without needing to define their structure as strongly. This also allows for the addition of more argument schemes in the future and is extremely extensible.

The database schema requires a large amount of relations between objects and, in a NoSQL database such as DynamoDB, this can be hard to maintain as a NoSQL database does not naturally manage foreign key relationships between objects. Specific unit tests were designed to ensure that foreign key relationships are maintained when performing database manipulations,

safeguarding against such a concern.

#### 4.2.2 Endpoints

Argupedia has several published endpoints which are used primarily to interact with the database. Each endpoint has its own handler which is responsible for verifying any authorisation information sent with a request, validating the request body or parameters, and performing the operations required to generate the correct output.

There are 23 endpoints in total and their details are outlined in Appendix A. For each published endpoint, the appendix shows the endpoint's path, any path parameters, the headers required while calling the endpoint, the request body required, all possible responses from the endpoint, and a description of what the endpoint does.

#### 4.2.3 Argumentation Schema Logic

The argumentation system described in the report background section has been implemented in this project. Due to the design of the overall application, there are a few reductions on the described argumentation system that can be made.

Firstly, arguments can only be created as a singleton or responded to either by an attacking argument or a supporting argument. There is no way in the current implementation of Argupedia for the user to create a relation between two existing arguments. An implementation of Argupedia was tested where a user could specify existing relations between arguments but the choice was made to use the current implementation as this is more user friendly and allows for the user to be better guided through the experience of creating and responding to arguments.

This means that all graphs of arguments within Argupedia can be considered trees and there should never be a cycle within an argument graph. Algorithms that analyse any argument graphs can use tree traversal.

This also means that there should never be multiple preferred labellings as no arguments should be able to attack each other. Any attacking arguments which join the tree can be added as IN and any supporting arguments which are added to the tree can be labelled the same as their parents.

Lastly, this reduction means that the only possible way for an argument to be undecided is for one of the argument's attackers to be labelled undecided. Recalling the definition of an undecided argument,

**Remark.** *x is legally UNDEC iff x is labelled UNDEC and not every y that attacks x is labelled OUT, and there is no y that attacks x such that y is labelled IN*

If all relations between arguments are representable as a tree, no attacking argument is IN, and not all attacking arguments are OUT, then at least one child argument must be UNDEC.

The second major reduction relates to supporting arguments. The extension of the argumentation schema which allows us to use supporting arguments essentially considers supporting arguments to essentially act as a block of arguments. An attack on any of these arguments is an attack on all of them and they should all share the same labelling.

To maintain a tree schema for arguments, each argument must be able to have at most one parent. To maintain this, whenever an attacking argument is created, the database should always make the argument attack the root of the supporting argument tree. This way, any labelling formula can go down a supporting argument tree at any node and label all alike.

With these reductions, the following code for labelling argument can be used

```
const labelArgumentAndSupportingArguments = async (
  argumentId: string,
  label: 'IN' | 'OUT' | 'UNDECIDED'
) => {

  // Retrieve argument from database
  const argument = await getArgumentById(argumentId);

  // Set label of argument to provided label
  argument.label = label;

  // Store the argument back in the database
  await putArgument(argument);

  // For each child argument, label it the same as the parent
  for await (const supportedByArgumentId of argument.supportedByArgumentIds) {
    await labelArgumentAndSupportingArguments(supportedByArgumentId, label);
  }
};

const labelArguments = async (leafArgumentId: string) => {
  // Retrieve argument from database
  const argument = await getArgumentById(leafArgumentId);

  // Assume that the argument is in by default
  let argumentLabel: ArgumentLabel = 'IN';

  // If the argument supports an argument, label it the same as the argument it supports
  if (argument.supportsArgumentId !== undefined) {
    const supportingArgument = await getArgumentById(
      argument.supportsArgumentId
    );
    argumentLabel = supportingArgument.label;
  }

  // Set label of argument to provided label
  argument.label = argumentLabel;

  // Store the argument back in the database
  await putArgument(argument);
};
```

```

);
argumentLabel = supportingArgument.label;
}

// Go through all attackers of the argument
for await (const attackedByArgumentId of argument.attackedByArgumentIds) {
  const attackingArgument = await getArgumentById(attackedByArgumentId);

  // If an attacking argument is UNDECIDED, label the argument as UNDECIDED until an argument label
  if (attackingArgument.label === 'UNDECIDED') {
    argumentLabel = 'UNDECIDED';
  }

  // If any attacking argument is IN, label the argument OUT
  if (attackingArgument.label === 'IN') {
    argumentLabel = 'OUT';
    break;
  }
}

// Label any arguments which support the argument
await labelArgumentAndSupportingArguments(leafArgumentId, argumentLabel);

// Continue up the tree and label the argument's parent argument
if (argument.attacksArgumentId !== undefined) {
  await labelArguments(argument.attacksArgumentId);
}
};


```

The `labelArguments` function is called whenever a new argument is added to the tree and is called on the new argument being created. As this is a leaf, the function only needs to traverse up the tree re-labelling any parent arguments until the function reaches the root of the argument tree.

If an argument that the function reaches has children, the function labels the argument correctly and then calls `labelArgumentAndSupportingArguments`, which traverses down all supporting arguments and ensures that they are labelled alike.

#### 4.2.4 Backend Resources

Provisioning of backend resources is defined using the infrastructure as code framework. This ensures that the project can be completely deployed as submitted with little configuration necessary. It also ensures that there are no orphaned resources when making changes to the system and allows for easily managed deployment of multiple environments (for development and production). Infrastructure as code allows the backend configuration to be entered in source control and allows for the rollback to previous backend configurations if required. The serverless

framework being used for this project supports its own implementation of infrastructure as code and this is what the project uses to provision resources.

The backend resources provisioned by this schema are as follows:

- User Table
- Argument Table
- Comment Table
- Cognito User Pool
- Cognito User Pool Client
- API Gateway Authorizer
- Gateway Response Controller
- API Gateway

Serverless generates an AWS CloudFormation file and uploads it along with all required resources to AWS to be provisioned. The configuration file specifies how the resources can identify each other and how they should authenticate with each other on the backend. The script also provisions log streams in AWS CloudWatch so that any issues or errors that arise can be noted and handled appropriately. Lastly, Serverless handles performance monitoring and can notify the deployer when there is a fatal error.

### 4.3 Frontend

The frontend is written in React using TypeScript and makes extensive use of the Ionic framework.

React is an open-source SDK making state management and rendering of the state much simpler. As an extension of React's native state management, Argupedia makes use of Zustand which is built to solve some of the problems of React's native state manager including context loss and React concurrency. Additionally, React includes support for JSX which allows for components to be generated by functions and reduces duplication across the codebase.

Ionic is an open-source SDK that includes many useful components and styled elements that make it easier to build a visual interface. Implementing Ionic also covers website responsiveness and allows the site to function well on all screen sizes. Argupedia's frontend was designed for

display on a desktop web browser but through Ionic, the frontend works well on mobile and tablet devices as well. It is possible to also deploy fully contained mobile apps to Android and iOS through Ionic – Ionic handles the creation of all needed files required to ship apps on both platforms.

The frontend also makes use of AWS Amplify (AWS’s frontend tooling suite). Amplify has many features but Argupedia makes use of only two of them: hosting and authentication.

### 4.3.1 Hosting

Amplify has been connected to the frontend’s GitHub repository and any changes deployed to the master branch are automatically picked up by Amplify, are built in the cloud, and then are deployed and made accessible to the world wide web. Amplify also will deploy ”preview” sites for any pull requests so that, during development, the changes can be viewed and tested in a siloed environment.

### 4.3.2 User Authentication

Additionally, Amplify’s authentication module sits as a connection between the frontend and the AWS Cognito service in the backend. Argupedia makes use of the module’s user onboarding and sign-in components which are out-of-the-box and handle authentication on their own. Any components which need to access sensitive data can be wrapped in this module and, if there is no user signed in during render, the module will kick the user over to these components to get their authentication details. The module can then be accessed from within the code to generate tokens for the signed-in user which are needed to make calls to protected endpoints. Using an Amplify authentication with Cognito ensures that user data is protected as these software are already compliant with relevant data protection regulations and standards and all PII is saved through these services.

### 4.3.3 Navigational Layout

The site is split into several distinct pages which are discussed in the next section. Users can navigate through the main pages by clicking on the hamburger icon in the upper left-hand corner which reveals a side panel containing links to the main pages.

#### 4.3.4 Pages

##### Home Page

The homepage is the page that is first served when the user logs in. It shows the user the newest arguments that have been created on the site. Clicking on any of these arguments takes the user to the page for that argument.

##### Argument Page

Each argument can be viewed on its argument page. The argument page has three major sections – a section to view the argument details, a section that visually represents the graph of arguments, and a section where users can see and create comments which correspond with the article.

In the argument details section, the user can see the text representation of the argument along with the argument's propositions. The user can also see when the argument was last modified. In this panel, the user can select whether they would like to respond to the argument with an attacking or supporting argument or, if the user is the one who created the argument, they can choose to edit the argument. Selecting these options takes you to the create argument or edit argument pages respectively.

In the graphical representation, the arguments are shown as a tree. Each argument node displays the argument's text and the user can jump between arguments by clicking on an argument within the tree. Arguments that are coloured green are arguments that have been labelled IN and arguments that are coloured red are arguments that have been labelled OUT. Additionally, green edges represent a supporting relation and red edges represent an attacking relation. If, for example, node A was attacking node B, Argupedia would draw a red line from note A to node B with an arrow pointing at node B. This representation is designed to give the user a good indication of the context that the currently viewed argument is in and also is a navigational aid, allowing the user to easily navigate arguments that are related to each other.

Lastly, in the comments section of the page, the user is shown all existing comments that have been created alongside when they were authored and who their author is. Clicking on the author's name takes the user directly to the author's profile where other arguments and comments the author has made can be seen. This section is intended to facilitate additional discussion about the argument.

## Create Argument Page

The create argument page handles new arguments, new supporting arguments, and new attacking arguments based on the URL path used to reach the page is. The page is designed to guide the user through creating a new argument and inspire their choices for how to respond.

If the user is creating an attacking or a supporting argument, the page first shows the user the existing argument so that the user can refer to it while crafting their new argument. The existing argument's argument text and propositions are both shown. The user is then shown several critical questions which are generated based on what argumentation scheme was used by the parent argument.

For example, if the user is responding to an argument using the appeal to expert opinion schema and the argument is "Dr Smith is an expert in virology and suggests that it is true that there is a pandemic", Argupedia would generate the following critical questions designed to guide the user's thinking:

- How credible is Dr Smith as an expert?
- Is Dr Smith an expert in the field that there is a pandemic is in?
- What did Dr Smith assert that implies there is a pandemic?
- Is Dr Smith personally reliable and trustworthy? Do we have any reason to think Dr Smith is less than honest?
- Is there is a pandemic consistent with what other experts assert?
- Is Dr Smith's evidence based on evidence?

The user is encouraged to use these questions when thinking about how they want to respond to the argument.

All users creating an argument are shown a form that handles the logic for the creation of the argument. Users are first prompted to select an argument schema from the list of supported argument schemas discussed in the background section of this report.

After the user has selected an argument type, the user is shown all the propositions of the argument. The user can either enter their values for the propositions or, for some propositions, the user can open a dropdown menu by selecting the plus icon on the right-hand side of the screen. The dropdown menu contains all propositions from the parent argument and clicking one of them will set the value of the new argument's proposition that has been selected. The

purpose of this functionality is to allow the user to more easily craft responding arguments as parts of responding arguments should be the same as their parents.

Lastly, the user is shown a text representation of their argument which is generated by Argupedia. The text representation is often completely correct but due to the intricacies of the English language, the text generated may be a little off from what is expected. The user can correct any errors in the automatic generation in this step by simply editing the text. This field is optional and the user does not have to modify the text at all. If the user leaves the text as-is, the automatically generated text representation of the argument is used.

Once the user clicks the submit button, they are taken to the new argument's page.

### **Edit Argument Page**

As discussed in the previous section, the automatically generated text representation of an argument may not always be grammatically correct. The edit argument page is designed to allow the author of the argument to change the text representation of their argument should this be the case or should the user feel that their argument is not properly represented by what they wrote previously.

The edit argument page consists of a simple form that shows the existing argument text and prompts the user to change the argument text as they wish. When the user clicks the "Save" button, their new argument text is saved and replaces the old argument text in the database.

### **Profile Page**

The profile page shows any user's basic details including their profile name and biography along with a list of arguments that they have authored and a list of comments that they have authored. Clicking on any of these arguments or comments will take the user to the argument page associated with the argument or comment. This will allow the user to view the argument or comment in context.

If the user is viewing their profile, they will be presented with the ability to edit the details of their profile by clicking an "Edit Profile" button. Doing so will take the user to the edit profile page.

### **Edit Profile Page**

The edit profile page allows the user to make changes to their biography and display name. The user may want to change these details to maintain anonymity on the platform or to update

their biography with new developments in their life.

The page consists of a form that shows the user's existing display name and biography and prompts the user to change them as they wish. The user can submit their changes by pressing the "Save" button at which point their updates will overwrite the existing values in the database for the user's display name and biography.

#### 4.3.5 API Access

To simplify calls to Argupedia's API, API calls are made through helper methods that structure requests to the backend, fetch and provide relevant authorisation data, and process the backend's response. The API methods that use authorisation data make a call to the AWS Amplify module to securely retrieve the tokens needed to make a request.

#### 4.3.6 Argument Translations

When the user is creating arguments or when the server is trying to display an argument to a user, a text representation of the argument is needed as this is the way that humans naturally process arguments. Argupedia has a robust helper method that generates text representations of the argument based on the argument's propositions.

The following example is a code snippet from the argument translations helper function and is a good representation of how Argupedia interprets and translates arguments into a more human format. The snippet converts an object representing an argument from position to know into a text representation of that argument:

```
case "ArgumentFromPositionToKnow": {
    const argumentFromPositionToKnow = argument as ArgumentFromPositionToKnow;

    return `${capitalizeFirstLetter(
        argumentFromPositionToKnow.person
    )} is in a position to know whether ${
        argumentFromPositionToKnow.fact
    } is true or false and they suggest that it is ${
        argumentFromPositionToKnow.proposition
    }.`;
}
```

The snippet inserts the propositions of the argument from position to know into the right positions of a string. There is code to convert every argument type inside of Argupedia into a text representation.

Argupedia relies on these representations to better represent the arguments to the end-user and this method is called throughout the platform wherever an argument needs to be shown to the user.

#### 4.3.7 Critical Question Generation

When the user is creating an attacking or supporting argument, they should be guided by Argupedia as they craft their argument. To get the user thinking about how to respond to an argument, Argupedia will generate and present several critical questions for the user to consider. There are specified questions that correspond to each argument schema and the questions are stored as strings that can have their text replaced.

As an example, the critical question strings for the argument from position to know are shown below:

```
ArgumentFromPositionToKnow: [
    "Is $person really in a position to know whether $fact is true? What is it about
     $person that makes them likely to know $fact?", 
    "Is $person an honest, trustworthy, and reliable source? Would $person have any
     reason to mislead?", 
    "Did $person really assert that $fact is $proposition? Are we hearing what $person
     said first-hand or second-hand? Is there reason to be suspicious about the
     fidelity of the information-transfer?", 
],
]
```

Each string contains substrings which start with the \$ character and indicate replaceable portions of the string. There is a helper method which is called when displaying critical questions which replaces text all relevant critical question strings:

```
const formatCriticalQuestions = (argumentType: string, argument: Argument) =>
  criticalQuestions[argumentType].map((question) => {
    for (const key of Object.keys(argumentTypeSchema[argumentType])) {
      question = question.replaceAll(`$$${key}`, (argument as any)[key]);
    }
    return question;
});
```

The output of this function is used by Argupedia in the create new argument page and is a critical part of helping the user navigate through the process of creating a new argument.

### 4.4 Software Engineering Practices

The production of Argupedia incorporated relevant software engineering practices to ensure code quality and organization.

#### **4.4.1 Waterfall**

The waterfall approach towards software development was a good fit for Argupedia. Scrum was considered with sprints between each project check-in meeting but many aspects of scrum would have been redundant as scrum and other methodologies like it look to solve problems that arise when working with others on software. For a single developer, the waterfall approach made sense.

The major stages for this project were somewhat standard for waterfall and included requirements elicitation, design, implementation, and verification. As this project must be submitted whole and work must stop after submission, there is no maintenance stage and there is no tooling as part of Argupedia for maintenance or follow-up release techniques such as A/B testing. Instead, the focus was placed on the testing stage to ensure that the software was behaving correctly and ready for submissions.

The check-in meetings for this project could be used to ensure that each step in the waterfall was being completed correctly. Early meetings focused on validating the design of Argupedia and ensuring that the suggested requirements were sufficient enough for the project. As progress was made along the waterfall, the meetings could focus on more acute aspects of the project.

There were, of course, changes to the project design or requirements as the project developed. Some initially proposed features were deemed infeasible for this project as their development began and some design aspects of the project changed from initial assumptions as the implementation phase elicited better solutions. In these situations, it was necessary to take a step back up the waterfall and re-evaluate the approach of the design. This is not typical for waterfall – a true waterfall approach requires completely moving on from one stage to another – but using this approach allowed for learnings from different stages to be incorporated into the final product.

#### **4.4.2 Kanban**

During the implementation and testing phases, a Jira Kanban board was used to track development progress on the project. Jira is a popular software development tool that can be used with myriad software engineering frameworks.

A ticket was created for each distinct part of the codebase which needed to be built and tickets fell into two epics – one for the backend and one for the frontend. The tickets detailed what needed to be built and allowed development to have a high-level view of how the project was coming along.

Use was made of Jira's integration with GitHub which allows the Kanban board to be connected with Argupedia's code repositories. Whenever work was started on a new issue, Jira would create a branch in the relevant repository on which work could be tracked by Jira. The results of any actions on that branch or any development on that branch would be reflected in the Kanban board and this allowed for the board to automatically update when development was taking place.

#### 4.4.3 Code Style

To enforce code style for the project, a linter and a formatter are employed. Code committed to the main repositories must meet formatting and linting rules first to be accepted as a pull request. Argupedia makes use of Prettier for formatting and ESLint for linting – both are popular choices for TypeScript projects and ensure that the code is highly readable and has consistent formatting.

#### 4.4.4 Continuous Integration

Code for the backend is stored in a private GitHub repository. To ensure that the repository history is well organized, all commit messages make use of the conventional commits specification and a new branch is made for every feature, bug, or issue. Code is never committed directly to the master

Two automated actions are also configured in GitHub – one for testing pull requests and one for deploying changes merged to the master branch.

The testing action performs the following actions:

- Download latest changes
- Install required software dependencies
- Run a formatting check to ensure that the code meets the code style guide
- Run a lint check to ensure that the code meets style rules
- Run all unit tests to ensure that any new changes have not broken any existing parts of the code
- Generate a code coverage reports which indicates whether the new code is sufficiently covered by unit tests
- Run SonarCloud to ensure that the code meets quality and security standards.

These actions ensure that any potential code is safe and meets the project standards. If all checks pass, the code is allowed to be merged with the master branch.

The deployment action performs the following actions:

- Download latest changes
- Install required software dependencies
- Generate a code coverage reports which indicates whether the new code is sufficiently covered by unit tests
- Run SonarCloud to ensure that the code meets quality and security standards.
- Run Serverless to deploy new changes to AWS and update any resources in the cloud.

This action is run on any pushes to the master branch and is more focused on release management. Serverless automatically identifies which resources need to be changed and updates them in this action.

SonarCloud analysis is generated for changes to the master branch as this allows sonar to generate a report for the entire repository, as opposed to just individual pull requests as it is doing in the testing action.

In both actions, GitHub needs to authenticate with SonarCloud and with AWS via Serverless to post new analyses and update cloud resources respectively. The tokens and credentials required to perform these actions are managed by GitHub as encrypted secrets. The AWS keys are particularly important as they provide access to update, manage, and provision resources on the AWS account where the repository is managed. As such, there is only one set of keys that can perform these actions and these keys are only live in the secrets manager. Cloud resources are therefore protected: the only way to change or modify resources in the cloud is to have the master AWS credentials or to commit to the master branch of the GitHub repository – access to which is managed.

On pull requests and commits, a GitHub Action is run all of the previously specified linting checks, formatting checks, security checks, and unit testing to ensure that the commit or pull request is safe to push or merge. The GitHub action will also manage deployment to AWS.

## **Chapter 5**

# **Legal, Social, Ethical and Professional Issues**

### **5.1 Legal Issues**

Argupedia must conform to the data protection laws of countries and regions where it may be accessed. The most stringent regulations with which argupedia may reasonably be expected to adhere is the European Union's GDPR restrictions which apply to European Union citizens and data that is transited through the European Union. GDPR laws have been duplicated in the United Kingdom, where this project is hosted, will be reviewed, and has had much of its work completed and California, where other significant parts of this project were completed, also has implemented the California Consumer Privacy Act which is also largely based on GDPR.

To ensure that Argupedia is compliant with these rules, data stored in Argupedia is stored in GDPR compliant areas. DynamoDB and AWS Cognito, where platform data and sensitive user data are stored, both conform to GDPR restrictions as well as the ISO 27001 standard which is an industry-standard for information security.

Through the use of these platforms and through implementing best practices for them to communicate with the frontend and with each other, any issues that arise from data protection laws are mitigated.

## **5.2 Social Issues**

Any consumer-facing platform which facilitates open discussion and discourse can potentially have users submit offensive content to the site. In the UK, offensive speech online can lead to prosecution and it is the responsibility of the user to regulate their speech online. At the moment there are no legal obligations for platforms themselves to moderate speech but this is likely to change in the coming months.

Argupedia seeks to be a platform that allows users to engage freely in online discussion and as such, seeks to promote constructive and non-offensive discourse by guiding the creation of arguments and keeping users focused on the discussion at hand.

## **5.3 British Computing Society Code of Conduct**

The standards outlined in the British Computing Society Code of Conduct were adhered to throughout the production of Argupedia. Argupedia poses no significant risk to the public or the environment other than those mentioned above. Argupedia was developed with due care for end-users and third parties.

# Chapter 6

# Results/Evaluation

Argupedia does not contain many runtime-complex algorithms nor does it require intense loads. Argupedia is a consumer-facing app and, throughout production, made use of industry-standard tooling to ensure that the codebase consisted of high-quality, working code.

To accomplish this, Argupedia used a quality evaluation tool for the frontend and backend, and a unit testing framework for backend methods. These two tools will be discussed further in this section and ensure that Argupedia is production-ready.

## 6.1 Quality Evaluation

The overall project makes use of SonarCloud to evaluate the quality of the code committed to the repository. SonarCloud performs automated code reviews and looks for bugs and security vulnerabilities within all submitted code.

SonarCloud is configured to run on both the frontend and backend repositories and is the main tool used for this project to evaluate the quality of the project.

SonarCloud produces scores in reliability, maintainability, security, and duplications. SonarCloud can also manage code coverage when configured to run with a unit tester. Coverage will be discussed in the Software Testing section of this chapter.

This section will examine the SonarCloud analysis on the backend and the frontend.

### 6.1.1 Backend

Argupedia's backend achieves the highest possible scores concerning SonarCloud's major evaluation categories as shown in figure 6.1. SonarCloud shows no bugs, vulnerabilities, or security

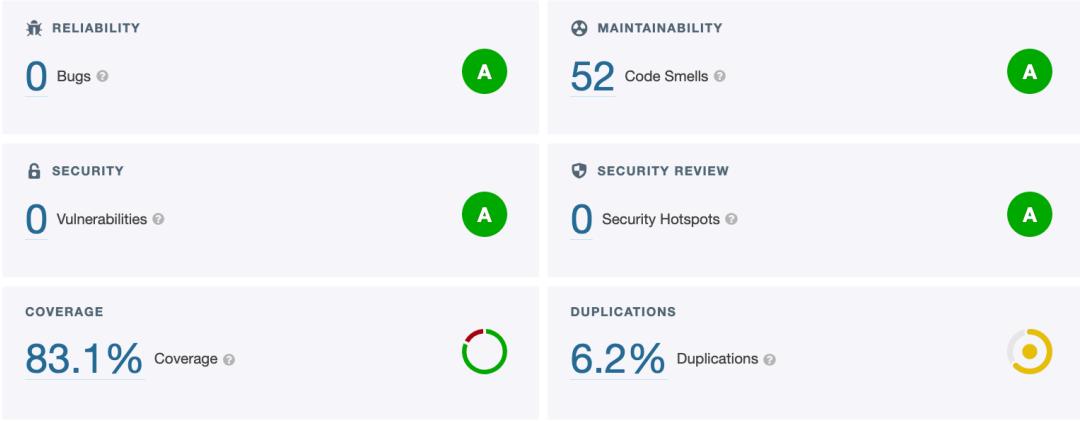


Figure 6.1: SonarCloud Analysis for Argupedia Backend

hotspots within the codebase.

Code smells are small characteristics of a codebase that may not inherently be problematic but may represent a bigger problem with the code. There are a total of 52 code smells but none are blocking and there is only one critical code smell. Upon further investigation, this critical code smell has to do with the complexity of the `validateArgument` method which is used by handlers in the backend to ensure that the argument and the argument type that a user is providing match. SonarCloud asserts that this file is too complex but this stems from the formatting of the file which causes the function to expand to include many lines thus increasing the cognitive complexity of the function. This code smell can be therefore dismissed as a false positive.

Overall the codebase has moderate duplications. This typically indicates that code should be refactored as the same code is being used in multiple places. Argupedia's backend was written with a middle layer which is designed to capture common backend operations thus reducing overall duplications so it is surprising to see an overall duplication rate of 6%.

Upon further investigation, it is clear to see that most of the duplication in the project comes from one file which is the same `validateArgument` function that has produced abnormal results for the aforementioned code smell metric. This file is represented as the outlier data point in figure 6.2 and this file has many duplications as the formatter used in the project has caused many single lines to be introduced in the file which are identical. These lines typically have opening or closing braces and SonarCloud has identified these lines as duplicates of each other. Disregarding this file, it is clear to see that the backend has an extremely low rate for duplications.

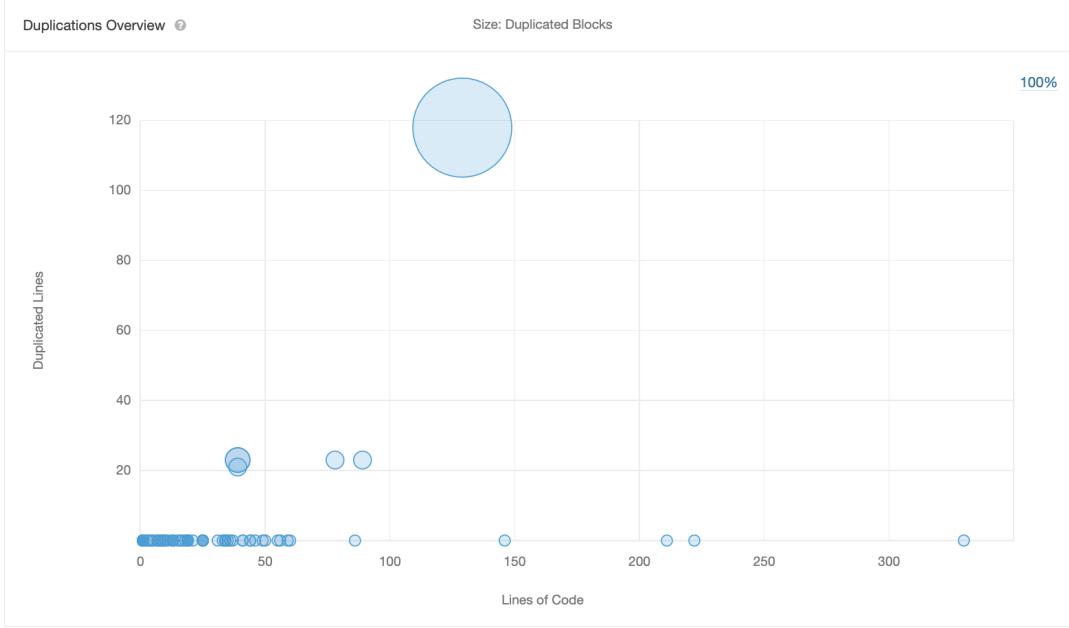


Figure 6.2: SonarCloud Duplication Analysis for Argupedia Backend

### 6.1.2 Frontend

Similar to the backend, Argupedia achieves the highest possible scores for the frontend as shown in figure 6.3. There are similarly no bugs, vulnerabilities, or security hotspots. The 70 code smells generated on the frontend have no critical code smells among them and the overall project has 5% duplicated code.

One of react's strengths is its ability to keep duplications low through the use of functional components which can allow the user to group commonly used elements. This is one of the ways that the frontend should manage to keep duplications low so a 5% rate for duplications is concerning. Looking at figure 6.4, only a few files are responsible for this duplication and these files, much like the file that has caused issues in the backend, are heavily mutated by the formatter and thus have many duplicated lines among them.

The frontend is thus well written and ready for production.

## 6.2 Software Testing

### 6.2.1 Backend

The backend makes use of extensive unit testing to ensure that all aspects of the backend are behaving as they should. Unit testing was completed using Jest which is a popular JavaScript

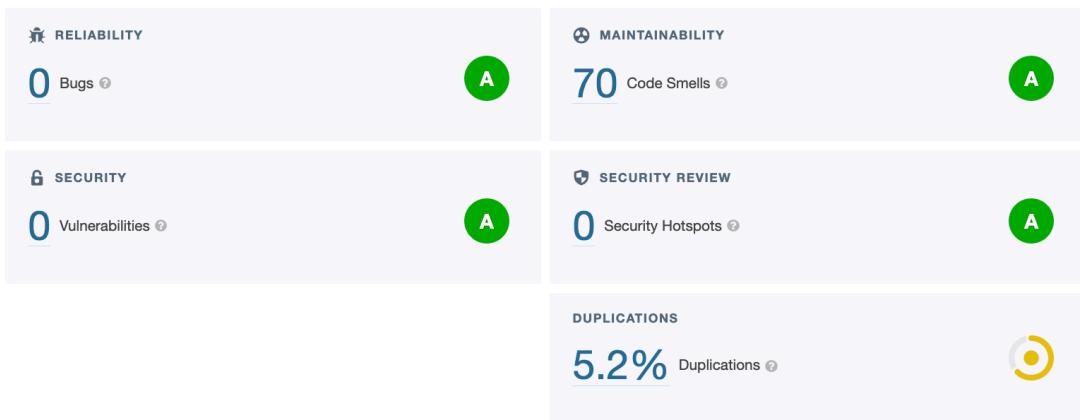


Figure 6.3: SonarCloud Analysis for Argupedia Frontend

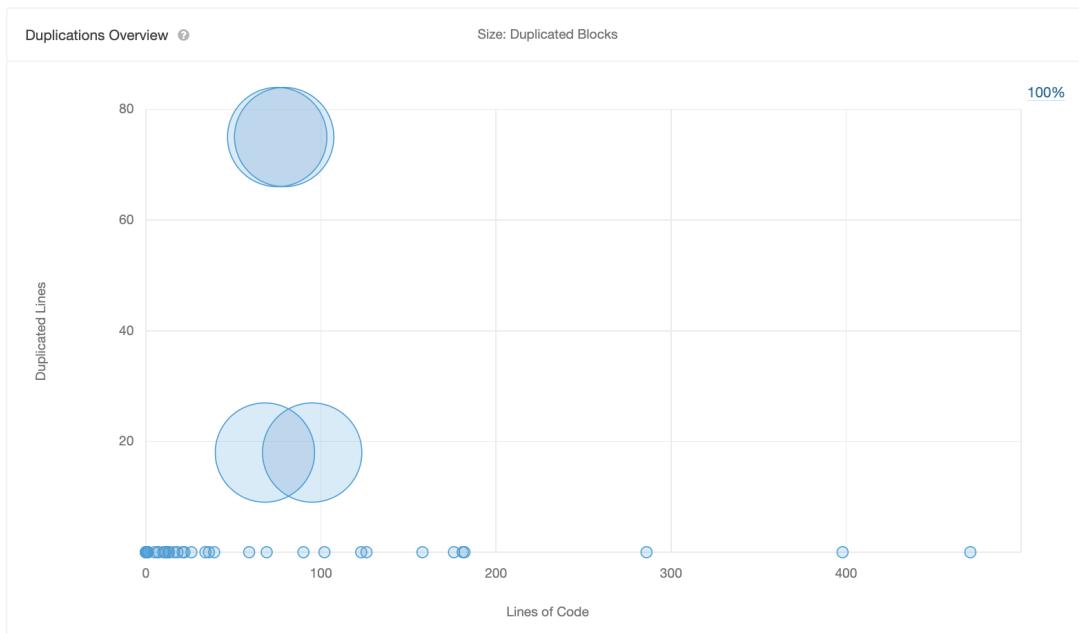


Figure 6.4: SonarCloud Duplication Analysis for Argupedia Frontend

t/TypeScript testing framework.

Jest was configured to mock calls to the database by creating a local instance of DynamoDB and making calls directly to that database. This was useful to ensure that the correct data was being written to the database, the database could be queried directly and having a local database ensured that the production database would not be disturbed or contaminated with data from testing.

To ensure that tests are as realistic as possible, Faker, a JavaScript library for generating realistic testing data, was used. A custom extension was built on top of Faker which allows the unit tests to generate fake database entries, fake arguments, and other fake objects relevant to Argupedia's typical use cases. The extension uses faker's realistic data generation to ensure that the objects used in unit tests are realistic. This ensures that as unit tests are run frequently during development, it can be reasonably guaranteed that the database works in all conditions.

To merge code to the master branch and subsequently push code into production, all unit tests must pass.

There are 96 unit tests covering a total of 28 files which represent the core functionality of the backend.

Jest generates a coverage report for the codebase which can be processed by SonarCloud to generate coverage metrics.

Overall, the project has achieved 83% coverage, shown in figure 6.1 for unit testing. This is good coverage and surpasses the 80% threshold suggested by SonarCloud. Looking more closely at the SonarCloud analysis, it is clear that a select few files account for most of the 20% of uncovered code. The largest of these files again contains the `validateArgument` function on which the formatter has left many lines of untestable code.

The backend is well tested and, during the implementation phase of Argupedia, the unit tests ensured that any new code being produced did not interfere with existing code.

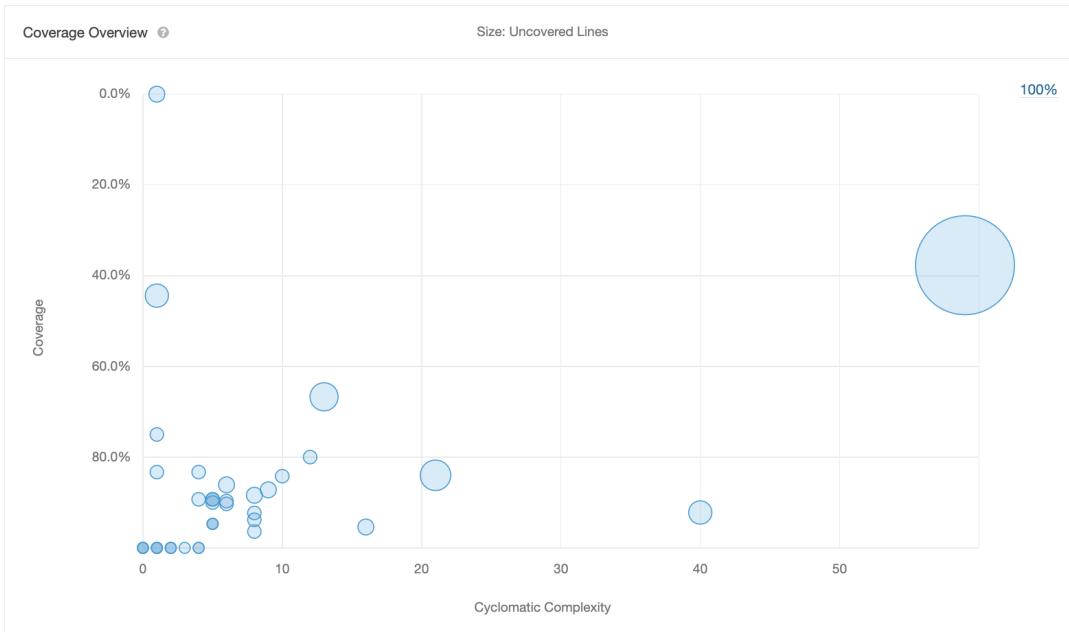


Figure 6.5: SonarCloud Coverage Analysis for Argupedia Backend

# Chapter 7

# Conclusion and Future Work

This project successfully deployed a full-stack application that allows users to create structured arguments with each other using an argumentation system.

## 7.1 Future Work

The final product works well and meets the original goals of the project but there is always more room for improvement. There are three major areas where Argupedia could be modified or improved: allowing the user to link existing arguments together, using user voting to resolve ungrounded arguments, and suggesting full arguments that the user can create based on the critical questions generated.

### 7.1.1 Linking Arguments

In the current design and implementation of Argupedia, arguments are represented as a tree as users can only ever respond to arguments with a new one and cannot create a relationship between two existing arguments. This is a design choice as it is believed that this will allow users to have a simpler time responding to and engaging in arguments: users do not have to think about the overall context of their new argument, they simply need to think about how their argument fits in with the existing ones.

This does not need to be the case, however, and Argupedia could be designed to allow users to link existing arguments together, thus creating additional edges on the argument graph. This design choice would allow users to create more complicated argument graphs but would require changes to the algorithms implemented to account for argument graphs that are not

necessarily trees.

### 7.1.2 Voting to Resolve Ungrounded Arguments

Assuming that Argupedia were to be augmented to allow users to link existing arguments together, argument graphs with no grounded extension. Argupedia's backend already can handle user voting: endpoints in the backend are exposed that allow users to upvote or downvote an argument. This is not implemented currently in the frontend but could be used to resolve a graph where two arguments prevent each other from reaching a grounded state. The argument which has a more positive score could be considered in, making the other argument out.

### 7.1.3 Suggesting Arguments Based on Critical Questions

Lastly, users are shown critical questions while they are creating new arguments to help them guide their thoughts but this concept could be expanded further. Argupedia could generate arguments based on the critical questions and the propositions from the parent argument could simply be inserted into the correct place. The user would only be required to enter any propositions not derived from the parent argument.

Implementing this feature would require a map to be created linking every critical question in Argupedia to a list of corresponding argument schemas.

# References

- [1] John Danaher. Argumentation schemes. *Philosophical Disquisitions*.
- [2] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.
- [3] Martin Caminada Sanjay Modgil. Action-based alternating transition systems for arguments about action. *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, 1:24–29, 2007.
- [4] Martin Caminada Sanjay Modgil. Proof theories and algorithms for abstract argumentation frameworks. *Artificial intelligence*, 2009.
- [5] Douglas Walton. *Fundamentals of Critical Argumentation*.

# Appendix A

# Argupedia API Documentation

This appendix includes the details of Argupedia's published API endpoints. For each published endpoint, the appendix shows the endpoint's path, any path parameters, the headers required while calling the endpoint, the request body required, all possible responses from the endpoint, and a description of what the endpoint does.

## A.1 Endpoints

### A.1.1 clearvoteArgumentWithId

Removes a user's upvotes or downvotes from the argument with the provided argument ID. The user's UUID is taken from the JWT provided in the authorisation header.

Path: *rest/api/1/argument/{argumentId}/clearvote (POST)*

Path Parameters:

- *argumentId*: The ID of the argument from which the user's votes should be removed.

Headers:

- *Authorization*: JWT token, authenticates the user making the request

Request body:

None

Response:

201: Returns the modified argument table entry

Error responses:

- 502  
No argument with specified argumentId.  
Returned when the argumentId provided is invalid.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.2 createCommentForArgumentWithId

Creates a new comment associated with a provided argumentId.

Path: `/api/1/argument/{argumentId}/comments (POST)`

Path Parameters:

- *argumentId*: The ID of the argument which the new comment should be associated with.

Request body:

```
{  
    commentBody: string // The text of the comment that should be created  
}
```

Headers:

- *Authorization*: JWT token, authenticates the user making the request
- *Content-Type*: application/json

Response:

201: Returns the created comment table entry

Error responses:

- 502  
No argument with specified argumentId.  
Returned when the argumentId provided is invalid.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.3 createNewArgument

Creates a new argument.

Path: `rest/api/1/argument/ (POST)`

Request body:

```
{  
    argumentType: string, // The name of the argument schema which is being used  
    argument: Argument, // The object representing the argument schema used  
    argumentText: string // (optional) A text representation of the argument being created.  
}
```

Headers:

- *Authorization*: JWT token, authenticates the user making the request
- *Content-Type*: application/json

Response:

201: Returns the created argument table entry

Error responses:

- 502  
argument does not match argumentType  
Returned when the argument object provided does not match up with the argument type provided.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

#### A.1.4 createNewAttackingArgument

Creates a new argument that is attacking an existing argument.

Path: *rest/api/1/argument/{argumentId}/attacking* (*POST*)

Path Parameters:

- *argumentId*: The ID of the argument which is being attacked.

Request body:

```
{
  argumentType: string, // The name of the argument schema which is being used
  argument: Argument, // The object representing the argument schema used
  argumentText: string // (optional) A text representation of the argument being created.
}
```

Headers:

- *Authorization*: JWT token, authenticates the user making the request
- *Content-Type*: application/json

Response:

201: Returns the created argument table entry

Error responses:

- 502  
No argument with specified argumentId.  
Returned when the argumentId provided is invalid.
- 502  
argument does not match argumentType  
Returned when the argument object provided does not match up with the argument type provided.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

#### A.1.5 createNewSupportingArgument

Creates a new argument which is supporting an existing argument.

Path: *rest/api/1/argument/{argumentId}/supporting* (*POST*)

Path Parameters:

- *argumentId*: The ID of the argument which is being supported.

Request body:

```
{  
    argumentType: string, // The name of the argument schema which is being used  
    argument: Argument, // The object representing the argument schema used  
    argumentText: string // (optional) A text representation of the argument being created.  
}
```

Headers:

- *Authorization*: JWT token, authenticates the user making the request
- *Content-Type*: application/json

Response:

201: Returns the created argument table entry

Error responses:

- 502  
No argument with specified argumentId.  
Returned when the argumentId provided is invalid.
- 502  
argument does not match argumentType  
Returned when the argument object provided does not match up with the argument type provided.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.6 createUser

Creates a new user object in the database which stores information such as the user's display name and biography. This endpoint does not create a new user in Cognito.

Path: *rest/api/1/user (POST)*

Request body:

```
{  
    biography: string, // (optional) The user's biography  
    displayName: string, // (optional) The preferred display name of the user  
    profilePictureURL: string // (optional) A URL to where the profile picture of the user is stored  
}
```

Headers:

- *Authorization*: JWT token, authenticates the user making the request
- *Content-Type*: application/json

Response:

201: Returns the created user table entry

Error responses:

- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.7 deleteArgumentWithId

Deletes the argument with the provided argumentId from the database.

Path: `rest/api/1/argument/{argumentId}` (`DELETE`)

Path Parameters:

- *argumentId*: The ID of the argument which is being deleted.

Headers:

- *Authorization*: JWT token, authenticates the user making the request

Response:

201: No data returned

Error responses:

- 502  
No argument with specified argumentId.  
Returned when the argumentId provided is invalid.
- 502  
The user is not the author of the specified argument  
Returned when the user that is trying to modify the argument is not the author of that argument.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.8 deleteCommentWithId

Deletes the comment with the provided commentId from the database.

Path: `rest/api/1/comment/{commentId}` (`DELETE`)

Path Parameters:

- *commentId*: The ID of the comment which is being deleted.

Headers:

- *Authorization*: JWT token, authenticates the user making the request

Response:

201: No data returned

Error responses:

- 502  
No comment with specified commentId.  
Returned when the commentId provided is invalid.
- 502  
The user is not the author of the specified comment  
Returned when the user that is trying to modify the argument is not the author of that argument.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.9 deleteUserId

Deletes the comment with the provided commentId from the database.

Path: `rest/api/1/user/{userId}` (`DELETE`)

Path Parameters:

- *userId*: The ID of the user which is being deleted.

Headers:

- *Authorization*: JWT token, authenticates the user making the request

Response:

201: No data returned

Error responses:

- 502  
No user with specified userId.  
Returned when the userId provided is invalid.
- 502  
User cannot update a different user  
Returned when the userId that is provided in the authorisation JWT header does not match the userId provided in the path parameter.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.10 downvoteArgumentWithId

Adds the user's userId to the list of downvoting userIds for an argument.

Path: `rest/api/1/argument/{argumentId}/downvote` (`POST`)

Path Parameters:

- *argumentId*: The argumentId which should be downvoted.

Headers:

- *Authorization*: JWT token, authenticates the user making the request

Response:

201: Returns the modified argument table entry

Error responses:

- 502  
No argument with specified argumentId.  
Returned when the argumentId provided is invalid.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.11 getAllArguments

Retrieves all arguments from the database.

Path: *rest/api/1/argument/* (GET)

Response:

201: Returns a list of all argument table entries on the site

### A.1.12 getArgumentGraphForArgumentWithId

Gets the argument graph that the specified argument is a part of.

Path: *rest/api/1/argument/{argumentId}/graph* (GET)

Path Parameters:

- *argumentId*: An argumentId within the graph that is requested

Response:

201: Returns a list of all argument table entries within the graph

Error responses:

- 502  
No argument with specified argumentId.  
Returned when the argumentId provided is invalid.

### A.1.13 getArgumentWithId

Gets the argument with the specified argumentId.

Path: *rest/api/1/argument/{argumentId}* (GET)

Path Parameters:

- *argumentId*: The argumentId of the requested argument

Response:

201: Returns the argument table entry of the argument with the specified argumentId.

Error responses:

- 502  
No argument with specified argumentId.  
Returned when the argumentId provided is invalid.

### A.1.14 getAuthoredArgumentsForUserWithId

Gets all argument table entries authored by the user with the provided userId.

Path: *rest/api/1/user/{userId}/authoredArguments* (GET)

Path Parameters:

- *userId*: The userId with which all arguments must be authored.

Response:

201: Returns a list of all argument objects authored by the user with the specified userId.

Error responses:

- 502  
No user with specified userId.  
Returned when the userId provided is invalid.

### A.1.15 getAuthoredCommentsForUserWithId

Gets all comment table entries authored by the user with the provided user ID.

Path: *rest/api/1/user/{userId}/authoredComments (GET)*

Path Parameters:

- *userId*: The userId with which all comments must be authored.

Response:

201: Returns a list of all comment objects authored by the user with the specified userId.

Error responses:

- 502  
No user with specified userId.  
Returned when the userId provided is invalid.

### A.1.16 getCommentsForArgumentWithId

Gets all comment table entries associated with the argument with the specified argumentId.

Path: *rest/api/1/argument/{argumentId}/comments (GET)*

Path Parameters:

- *argumentId*: The argument with which all comments must be associated.

Response:

201: Returns a list of all comment objects associated with the argument with the specified argumentId.

Error responses:

- 502  
No argument with specified argumentId.  
Returned when the argumentId provided is invalid.

### A.1.17 getCommentWithId

Gets the comment with the specified commentId.

Path: *rest/api/1/comment/{commentId} (GET)*

Path Parameters:

- *commentId*: The commentId of the requested comment

Response:

201: Returns the comment table entry of the comment with the specified commentId.

Error responses:

- 502  
No comment with specified commentId.  
Returned when the commentId provided is invalid.

### A.1.18 getUserId

Gets the user with the specified userId.

Path: *rest/api/1/user/{userId}* (GET)

Path Parameters:

- *userId*: The userId of the requested user

Response:

201: Returns the user table entry of the user with the specified userId.

Error responses:

- 502  
No user with specified userId.  
Returned when the userId provided is invalid.

### A.1.19 getVotedArgumentsForUserId

Gets all arguments that the user with the specified userId has voted on.

Path: *rest/api/1/user/{userId}/votedArguments* (GET)

Path Parameters:

- *userId*: The userId with which all arguments must be voted.

Response:

201: Returns a list of all argument objects voted on by the user with the specified userId.

Error responses:

- 502  
No user with specified userId.  
Returned when the userId provided is invalid.

### A.1.20 updateArgumentWithId

Updates an existing argument with new data.

Path: *rest/api/1/argument/{argumentId}* (PATCH)

Path Parameters:

- *argumentId*: The argumentId of the argument which should be updated.

Request body:

```
{
    argumentType: string, // (optional) The name of the argument schema which is being used
    argument: Argument, // (optional) The object representing the argument schema used
    argumentText: string // (optional) A text representation of the argument being created.
}
```

Headers:

- *Authorization*: JWT token, authenticates the user making the request
- *Content-Type*: application/json

Response:

201: Returns the updated argument table entry

Error responses:

- 502  
argument does not match argumentType  
Returned when the argument object provided does not match up with the argument type provided.
- 502  
No argument with specified argumentId  
Returned when the argumentId provided is invalid.
- 502  
User is not the author of the specified argument  
Returned when the user, identified by the credentials in the authorization header, is not the author of the specified argument.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.21 updateCommentWithId

Updates an existing comment with new data.

Path: *rest/api/1/comment/{commentId}* (PATCH)

Path Parameters:

- *commentId*: The commentId of the comment which should be updated.

Request body:

```
{
    commentBody: string // The text of the comment that should be created
}
```

Headers:

- *Authorization*: JWT token, authenticates the user making the request
- *Content-Type*: application/json

Response:

201: Returns the updated comment table entry

Error responses:

- 502  
No comment with specified commentId  
Returned when the commentId provided is invalid.
- 502  
The user is not the author of the specified comment  
Returned when the user, identified by the credentials in the authorization header, is not the author of the specified comment.
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.22 updateUserId

Updates an existing user with new data.

Path: *rest/api/1/user/{userId}* (*PATCH*)

Path Parameters:

- *userId*: The userId of the user which should be updated.

Request body:

```
{
  biography: string, // (optional) The user's biography
  displayName: string, // (optional) The preferred display name of the user
  profilePictureURL: string // (optional) A URL to where the profile picture of the user is stored
}
```

Headers:

- *Authorization*: JWT token, authenticates the user making the request
- *Content-Type*: application/json

Response:

201: Returns the updated user table entry

Error responses:

- 502  
No user with specified userId  
Returned when the userId provided is invalid.
- 502  
User cannot update a different user  
Returned when the user, identified by the credentials in the authorization header, does not match the userId specified in the path parameter
- 401  
Unauthorized  
The user has provided a bad token or is not authorised to access the endpoint.

### A.1.23 upvoteArgumentWithId

Adds the user's userId to the list of upvoting userIds for an argument.

Path: *rest/api/1/argument/{argumentId}/upvote* (*POST*)

Path Parameters:

- *argumentId*: The argumentId which should be upvoted.

Headers:

- *Authorization*: JWT token, authenticates the user making the request

Response:

201: Returns the modified argument table entry

Error responses:

- 502

No argument with specified argumentId.

Returned when the argumentId provided is invalid.

- 401

Unauthorized

The user has provided a bad token or is not authorised to access the endpoint.

## Appendix B

# Argument Schemas

The following are the argument schemas used within Argupedia. These are taken directly from the source code and are used by Argupedia to evaluate and process arguments. Each argument schema is represented by a JavaScript object where the keys of the object represent the argument schema's propositions and the values of the object represent the type for the proposition.

Within this appendix, the "consequenceType" type is an enum containing "GOOD" or "BAD".

## B.1 Argument Types used by Argupedia

### B.1.1 Action Argument

```
{  
    circumstance: "string",  
    action: "string",  
    newCircumstance: "string",  
    goal: "string",  
    value: "string",  
}
```

### B.1.2 Argument from Position to Know

```
{  
    person: "string",  
    fact: "string",  
}
```

```
    proposition: "boolean",  
}
```

### B.1.3 Appeal to Expert Opinion

```
{  
  
    expert: "string",  
    domain: "string",  
    fact: "string",  
    proposition: "boolean",  
}
```

### B.1.4 Appeal to Popular Opinion

```
{  
  
    fact: "string",  
    proposition: "boolean",  
}
```

### B.1.5 Argument from Analogy

```
{  
  
    originalCase: "string",  
    similarCase: "string",  
    fact: "string",  
    proposition: "boolean",  
}
```

### B.1.6 Argument from Correlation to Cause

```
{  
  
    cause: "string",  
    effect: "string",  
}
```

### B.1.7 Argument from Positive or Negative Consequences

```
{
```

```
        cause: "string",
        effect: "string",
        consequence: "consequenceType",
    }
```

### B.1.8 Slippery Slope Argument

```
{
    firstStepPremise: "string",
    recursivePremise: "string",
    consequence: "consequenceType",
}
```

### B.1.9 Argument from Sign

```
{
    finding: "string",
    fact: "string",
    proposition: "boolean",
}
```

### B.1.10 Argument from Commitment

```
{
    entity: "string",
    existingCommitment: "string",
    newCommitment: "string",
}
```

### B.1.11 Argument from Inconsistent Commitment

```
{
    entity: "string",
    commitment: "string",
    otherEvidence: "string",
}
```

### B.1.12 Direct Ad Hominem Argument

```
{  
    entity: "string",  
    criticism: "string",  
}
```

### B.1.13 Circumstantial Ad Hominem Argument

```
{  
    entity: "string",  
    originalArgument: "string",  
    otherEvidence: "string",  
}
```

### B.1.14 Argument from Verbal Classification

```
{  
    entity: "string",  
    property: "string",  
    classification: "string",  
}
```

## Appendix C

# Argument Critical Questions

### C.1 List of Critical Questions By Argument

#### C.1.1 Action Argument

- Is it true that *circumstance*?
- Assuming *circumstance*, does *action* bring about *newCircumstance*?
- Assuming the *circumstance* and that *action* will bring about *newCircumstance*, will *action* bring about *goal*?
- Does *goal* realise *value*?
- Are there alternative ways of realising *newCircumstance*?
- Are there alternative ways of realising *goal*?
- Are there alternative ways of promoting *value*?
- Does doing *action* have a side effect which demotes *value*?
- Does doing *action* have a side effect which demotes some other value?
- Does doing *action* promote some other value?
- Does doing *action* preclude some other action which would promote some other value?
- Is it possible that *circumstance*?
- Is the *action* possible?

- Is *newCircumstance* possible?
- Can *goal* be realised?
- Is *value* indeed a legitimate value?

### C.1.2 Argument from Position to Know

- Is *person* really in a position to know whether *fact* is true? What is it about *person* that makes them likely to know *fact*?
- Is *person* an honest, trustworthy, and reliable source? Would *person* have any reason to mislead?
- Did *person* really assert that *fact* is *proposition*? Are we hearing what *person* said first-hand or second-hand? Is there reason to be suspicious about the fidelity of the information-transfer?

### C.1.3 Appeal to Expert Opinion

- How credible is *expert* as an expert?
- Is *expert* an expert in the field that *fact* is in?
- What did *expert* assert that implies *fact*?
- Is *expert* personally reliable and trustworthy? Do we have any reason to think *expert* is less than honest?
- Is *fact* consistent with what other experts assert?
- Is *expert*'s evidence based on evidence?

### C.1.4 Appeal to Popular Opinion

- What evidence do we have for believing that *fact* is generally accepted?
- Even if *fact* is generally accepted as being true, are there good reasons for doubting its veracity?

### C.1.5 Argument from Analogy

- Is *fact* really *proposition* in *originalCase*?
- Are there differences between *originalCase* and *similarCase* that would tend to undermine the force of the similarity cited?
- Is there some other case that is also similar to *originalCase* but in which *fact* is *propositionOpposite*?

### C.1.6 Argument from Correlation to Cause

- Is there really a correlation between *cause* and *effect*?
- Is there any reason for thinking the correlation is anything more than a coincidence?
- Could there be some third factor that is causing both *cause* and *effect*?

### C.1.7 Argument from Positive or Negative Consequences

- How strong is the probability or plausibility that *effect* will occur?
- What evidence supports the claim that *effect* will occur?
- Are there consequences of not doing *cause* that ought to be taken into account?
- Is *cause* really *consequence*?

### C.1.8 Slippery Slope Argument

- What intervening propositions in the sequence linking *firstStepPremise* to *recursivePremise* are actually given?
- What other steps are required to fill in the sequence to make it plausible?
- What are the weakest links in the sequence, the places where key critical questions need to be asked?

### C.1.9 Argument from Sign

- What is the strength of the correlation between *finding* and *fact*?
- Are there other events that would more reliably account for *finding*?

### C.1.10 Argument from Commitment

- What evidence are we relying on to claim that *entity* is committed to *existingCommitment*?
- Is there room for questioning whether there is an exception in this case to the general rule that commitment to *existingCommitment* implies commitment to *newCommitment*?

### C.1.11 Argument from Inconsistent Commitment

- What is the evidence supposedly showing that *entity* is committed to *commitment*?
- What further evidence shows that *person* is not really committed to *commitment*?
- How does the evidence show that there is a conflict of commitments?

### C.1.12 Direct Ad Hominem Argument

- How well supported by evidence is *criticism*?
- Is *criticism* relevant in the dialogue in which the argument is made?
- Is the conclusion that *entity*'s argument should not be accepted absolute or merely one factor to consider when assessing the argument?

### C.1.13 Circumstantial Ad Hominem Argument

- Is *otherEvidence* reliable evidence?
- Could the practical inconsistency be resolved by further dialog? Should *entity* be given a chance to explain themselves?
- Is character a relevant consideration in this argumentative dialogue?
- What are we entitled to conclude? Can we ignore *originalArgument*? Or are we only entitled to ignore *entity*'s contributions to a more general debate?

### C.1.14 Argument from Verbal Classification

- What evidence is there for thinking that *entity* definitely is *property*? Are there reasons for thinking *entity* belongs to another category of objects?
- Is saying that those who are *property* probably *classification* a stipulative or biased definition that is subject to doubt?

# Appendix D

## User Guide

Argupedia has a simple, straightforward interface and is best evaluated by trying it out and exploring the features.

Argupedia is available to use at [argupedia.jonathandamico.me](http://argupedia.jonathandamico.me).

To build argupedia from scratch, follow the directions in the Installation & Setup section.

### D.1 Installation & Setup

The source code zip file contains two folders, `argupedia-api-master` and `argupedia-app-master`.

The first folder contains code for the backend and the second folder contains code for the frontend.

To deploy argupedia, an AWS account is needed. If you do not have an AWS account, an account can be created for free at [aws.amazon.com](http://aws.amazon.com).

Node.JS and NPM must be installed before starting. If Node.JS or NPM are not already installed on your machine, download the Node.JS installer from [nodejs.org/en/download](http://nodejs.org/en/download). To see if node installed on your machine, run `node -v` in your terminal. This should print the version number of node, if it is installed, or should error out, if node is not installed.

Argupedia's backend should be deployed first as details from this deployment are needed for the frontend deployment.

#### D.1.1 Backend Deployment

To deploy the backend, follow the following steps:

1. Open a terminal and navigate using `cd` to the directory where the backend code is stored.

This folder should be called `argupedia-api-master` and this folder will be in the root of wherever the source code for Argupedia has been downloaded.

2. Run `npm i` in the terminal.
3. Run `npm i -g serverless` in the terminal.
4. Generate AWS keys for deployment by following the following steps:
  - (a) Sign into the AWS console at `aws.amazon.com`, navigate to the Identity and Access Management (IAM) console by searching for "IAM" in the search bar and clicking the first result.
  - (b) Click on Users and then Add user. Enter a name in the first field to remind you this User is related to the Serverless Framework, like `serverless-admin`. Enable Programmatic access by clicking the checkbox. Click Next to go through to the Permissions page. Click on Attach existing policies directly. Search for and select AdministratorAccess then click Next: Review. Check to make sure everything looks good and click Create user.
  - (c) View and copy the API Key & Secret to a temporary place. They will be needed shortly.
5. Run the following command in the terminal, replacing `ACCESS-KEY-ID` and `SECRET-ACCESS-KEY` with the access key ID and secret access key from IAM.

```
serverless config credentials \
--provider aws \
--key ACCESS-KEY-ID \
--secret SECRET-ACCESS-KEY
```

6. Run `serverless deploy` in the terminal. This command should take awhile to run.
7. When serverless is finished deploying, make a note of the API url generated. This will appear in the format of `https://RANDOMCHARACTERS.execute-api.eu-west-2.amazonaws.com/dev`. This will be used while deploying the frontend. Take the URL from the highlighted area of the serverless deploy output in figure D.1.

After deploying the backend, move on to deploying the frontend.

```

Amazon WorkSpaces
Terminal

File Edit View Search Terminal Help
GET https://nlmpe89f.execute-api.eu-west-2.amazonaws.com/dev/rest/api/l/user/{userId}/votedArguments
GET https://nlmpe89f.execute-api.eu-west-2.amazonaws.com/dev/rest/api/l/argument/{argumentId}/graph
functions:
createNewArgument: argupedia-api-dev-createNewArgument
createNewAttackingArgument: argupedia-api-dev-createNewAttackingArgument
createNewSupportingArgument: argupedia-api-dev-createNewSupportingArgument
getArgumentWithId: argupedia-api-dev-getArgumentWithId
getAllArguments: argupedia-api-dev-getAllArguments
getAllAttackingArguments: argupedia-api-dev-getAllAttackingArguments
getAllSupportingArguments: argupedia-api-dev-getAllSupportingArguments
getVoteArgumentWithId: argupedia-api-dev-getVoteArgumentWithId
updateArgumentWithId: argupedia-api-dev-updateArgumentWithId
downvoteArgumentWithId: argupedia-api-dev-downvoteArgumentWithId
clearVoteArgumentWithId: argupedia-api-dev-clearVoteArgumentWithId
createCommentForArgumentWithId: argupedia-api-dev-createCommentForArgumentWithId
getCommentsForArgumentWithId: argupedia-api-dev-getCommentsForArgumentWithId
getCommentWithId: argupedia-api-dev-getCommentWithId
deleteCommentWithId: argupedia-api-dev-deleteCommentWithId
updateCommentWithId: argupedia-api-dev-updateCommentWithId
updateUserWithId: argupedia-api-dev-updateUserWithId
getUserWithId: argupedia-api-dev GetUserWithId
createUser: argupedia-api-dev-createUser
updateUserWithId: argupedia-api-dev-updateUserWithId
getAuthorizedArgumentsForUserWithId: argupedia-api-dev-getAuthorizedArgumentsForUserWithId
getAuthorizedCommentsForUserWithId: argupedia-api-dev-getAuthorizedCommentsForUserWithId
getVotedArgumentsForUserWithId: argupedia-api-dev-getVotedArgumentsForUserWithId
getArgumentGraphForArgumentWithId: argupedia-api-dev-getArgumentGraphForArgumentWithId
layers:
None
Serverless: Deprecation warning: Variables resolver reports following resolution errors:
- Cannot resolve variable at "provider.environment.USER_TABLE_NAME": Value not found at "self" source,
- Cannot resolve variable at "provider.environment.ARGUMENT_TABLE_NAME": Value not found at "self" source,
- Cannot resolve variable at "provider.environment.COMMENT_TABLE_NAME": Value not found at "self" source,
- Cannot resolve variable at "resources.Resources.CognitoUserPool.Properties.UserPoolName": Value not found at "self" source,
- Cannot resolve variable at "resources.Resources.CognitoUserPoolClient.Properties.ClientName": Value not found at "self" source,
- Cannot resolve variable at "resources.Resources.CognitoUserPoolClient.Properties.Name": Value not found at "self" source
From the next major this will be communicated with a thrown error.
Set "variablesResolutionMode: '20210126'" in your service config, to adapt to new behavior now.
More info: https://www.serverless.com/framework/docs/deprecations/#NEW_VARIABLES_RESOLVER

Monitor APIs by route with the Serverless Dashboard: run "serverless"
[jonathanhandamico\jonathan@o-ypgsn613863k argupedia-api-master]$ 

```

Figure D.1: URL Location in Serverless Deploy Output

### D.1.2 Frontend Deployment

To deploy the frontend, follow the following steps:

1. Open a terminal and navigate using `cd` to the directory where the frontend code is stored.  
This folder should be called `argupedia-app-master` and this folder will be in the root of wherever the source code for Argupedia has been downloaded.
2. Run `npm i` in the terminal.
3. Run `npm i -g @aws-amplify/cli` in the terminal.
4. Run `npm i -g ionic` in the terminal.
5. Run `echo "REACT_APP_API_BASE_URL={BACKEND_URL}" > ".env"` in the terminal. Replace `{BACKEND_URL}` with the URL captured in the backend deploy. This URL will follow the format of  
`https://RANDOMCHARACTERS.execute-api.eu-west-2.amazonaws.com/dev`. Be sure to include the `/dev` at the end.
6. Run `ionic serve` in the terminal. This will create a web server on localhost and make Argupedia accessible in the browser at `http://localhost:8100/`.

### D.1.3 Deleting Resources

To delete AWS resources, follow the following steps:

1. Open a terminal and navigate using `cd` to the directory where the backend code is stored.

This folder should be called `argupedia-api-master` and this folder will be in the root of wherever the source code for Argupedia has been downloaded.

2. Run `serverless remove`

All cloud resources should be removed and decommissioned.

## D.2 Application Use

### D.2.1 Creating an Account

When visiting Argupedia for the first time, a user account will need to be created.

After visiting Argupedia, the create account page, shown in figure D.2, will be displayed.

Enter an email account and choose a password on this screen.

After selecting "Create Account", a confirmation code will then be sent over email. The email will look like the email shown in figure D.4.

Enter the code from the email into the field shown in figure D.3 to create an account.

Once an account has been created, Argupedia will display the homepage, shown in figure D.6

### D.2.2 Editing your Profile

After creating a user account, it will still be devoid of a display name or biography.

To remedy this, edit the user account by clicking on the hamburger menu in the upper left-hand corner of the screen. An example is shown in figure D.6.

Clicking on the hamburger icon will reveal the side menu, shown in figure D.7. Select "View Your Profile" to navigate to the profile page, shown in D.8.

From the profile page, select "Edit Profile" to update user information. This will load the edit profile page, shown in figure D.9. Fill in a value for display name and biography, as shown in figure D.10 and click "Save"

Argupedia will now show the profile page with the most recent changes.

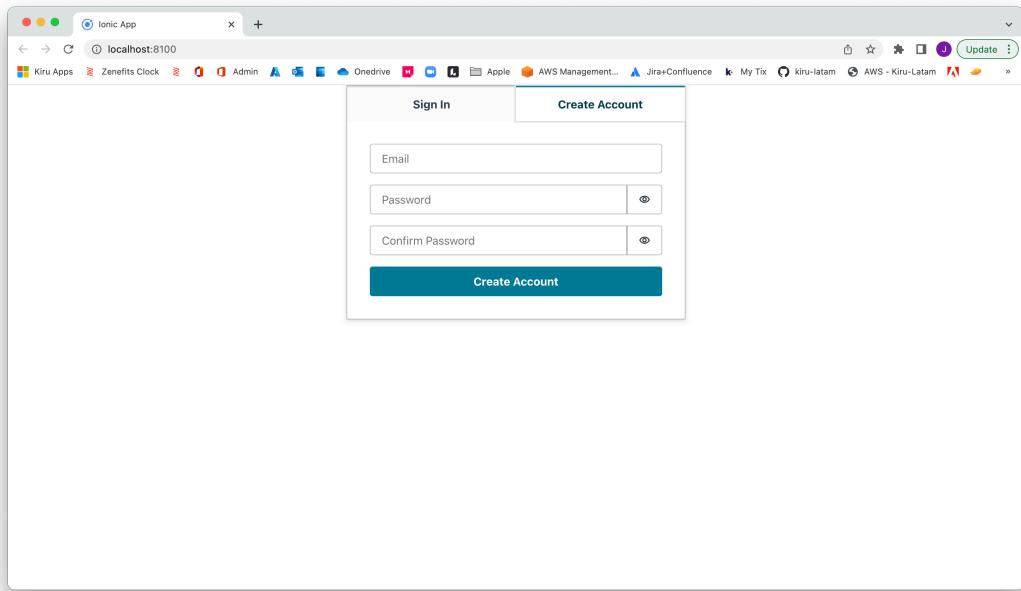


Figure D.2: Create Account Page

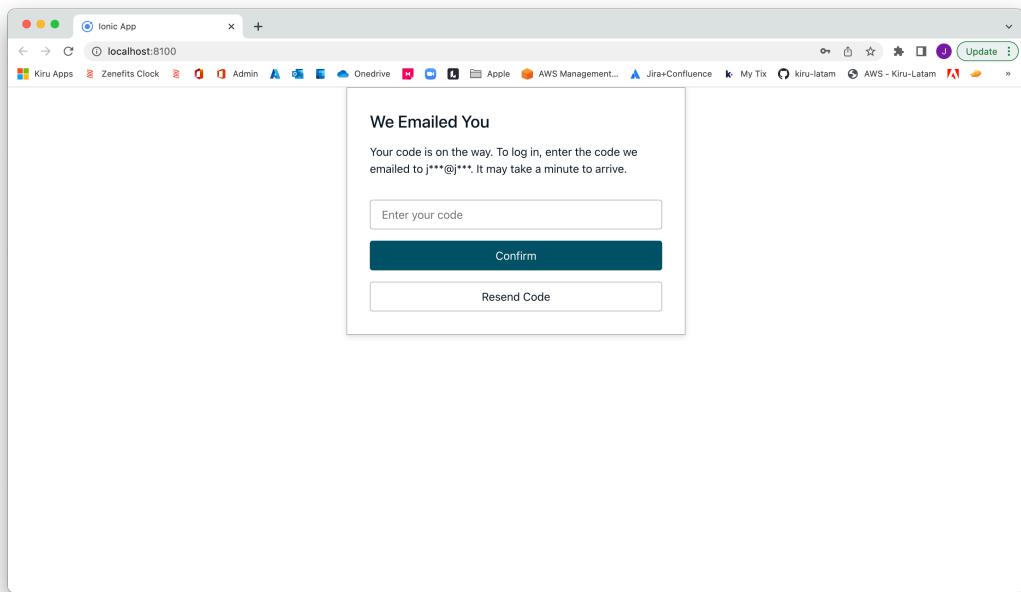


Figure D.3: Confirmation Code Request Page

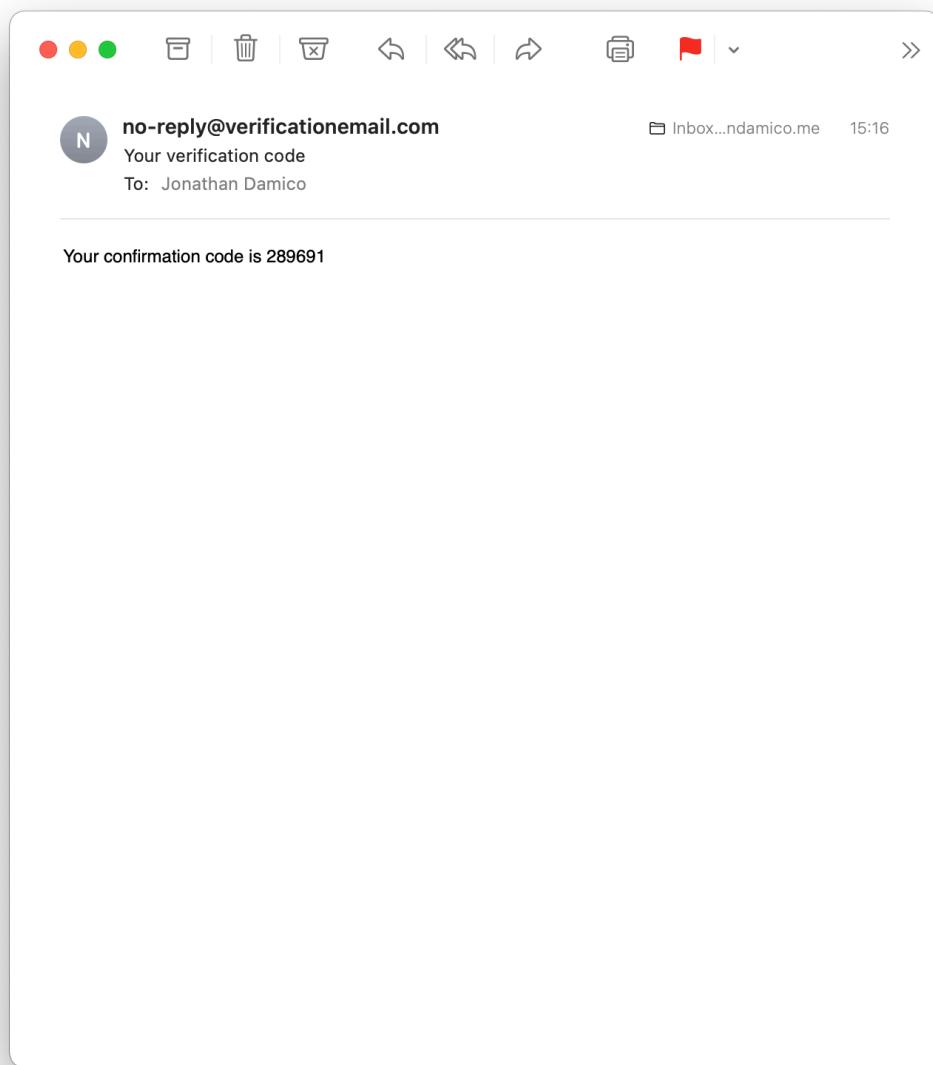


Figure D.4: Email with Confirmation Code

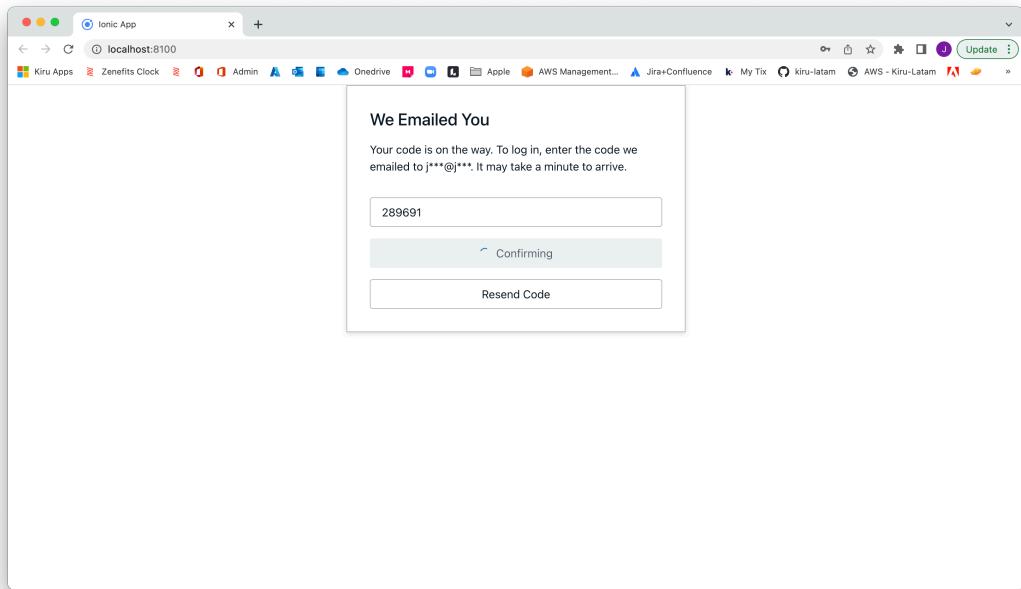


Figure D.5: Confirmation Code Request Page after Code Entry

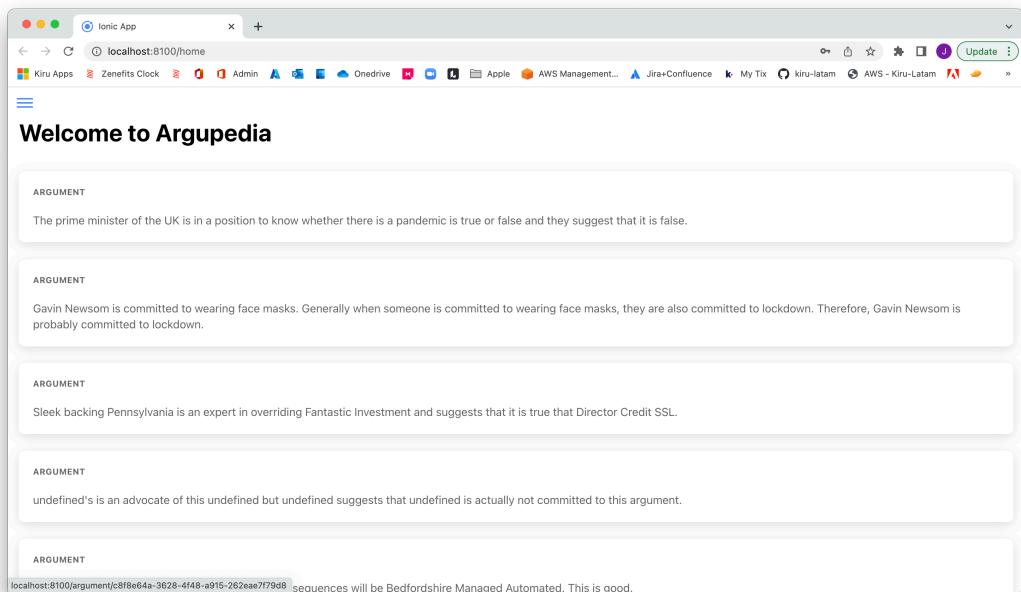


Figure D.6: Argupedia Home Page

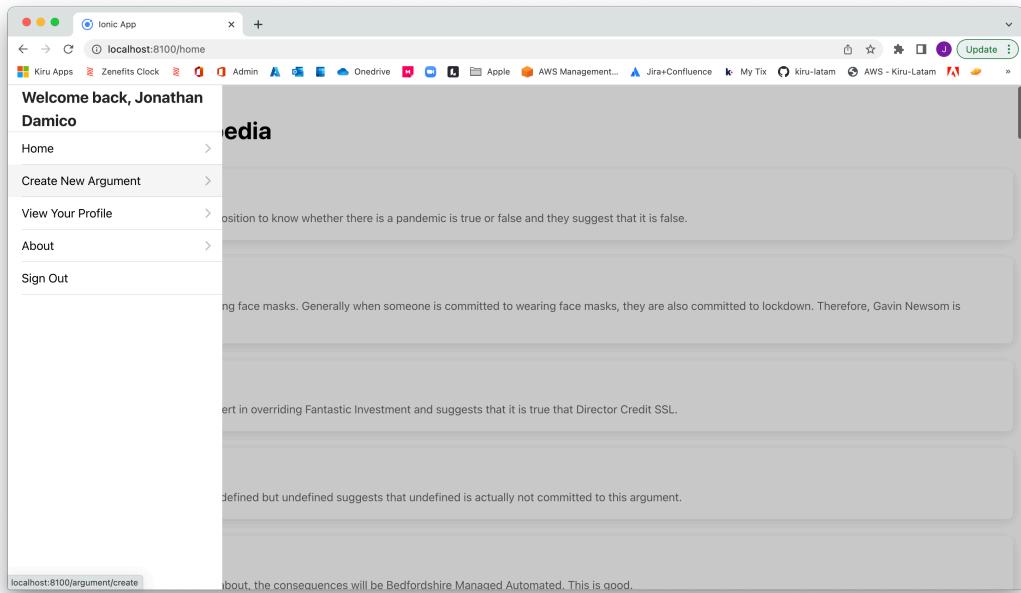


Figure D.7: Argupedia Side Menu

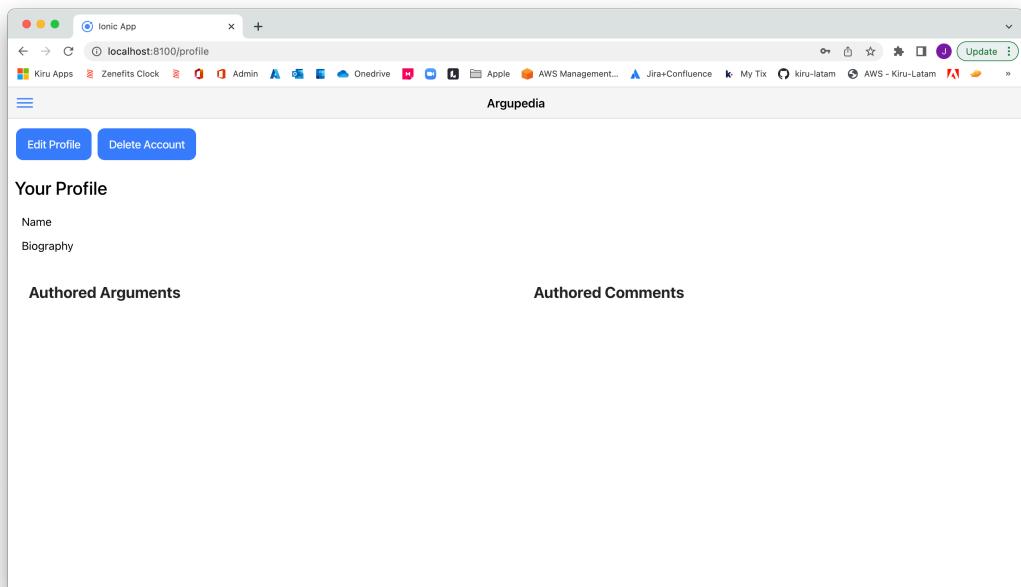


Figure D.8: Argupedia Blank User Profile

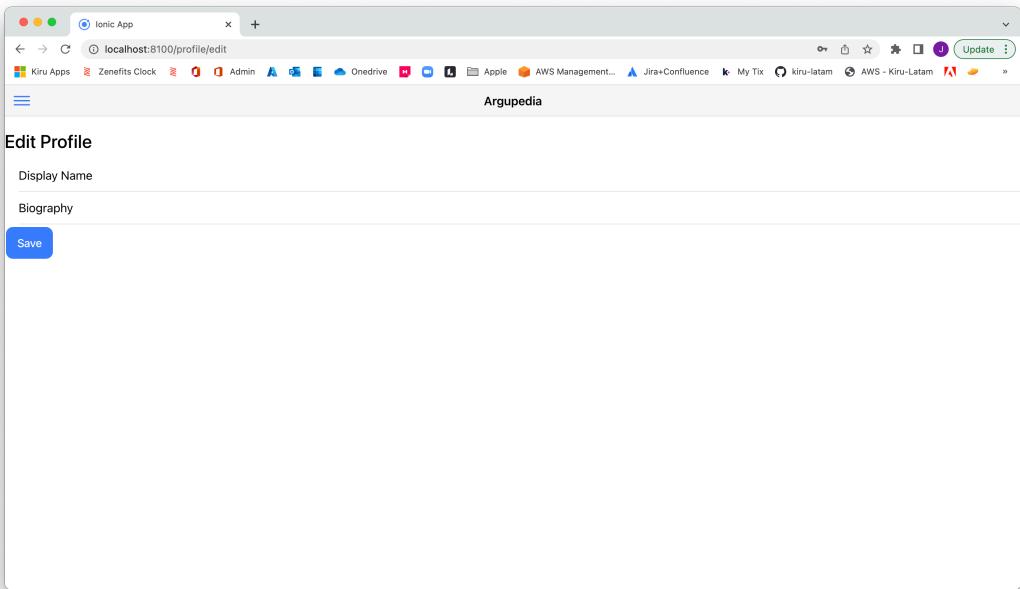


Figure D.9: Argupedia Edit Profile Page

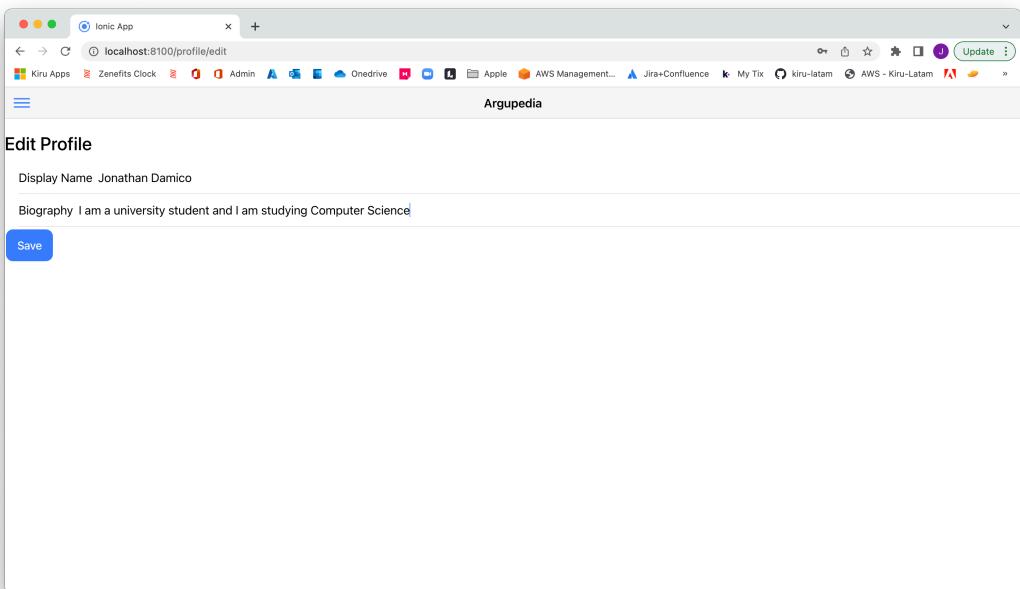


Figure D.10: Argupedia Filled-in Edit Profile Page

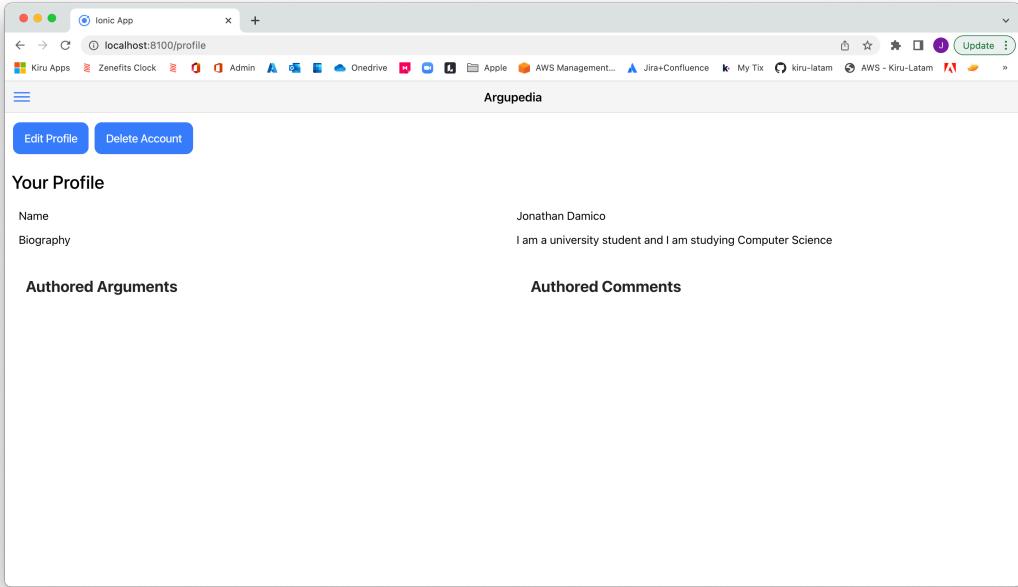


Figure D.11: Argupedia Filled-in Profile Page

### D.2.3 Creating an Argument

To create a new argument, navigate to the side menu by selecting the hamburger icon in the top left corner. From the side menu, shown in D.7, select "Create New Argument". This will load a page where a new argument can be created, shown in D.12.

On the create argument page, select an argument type from the dropdown menu, shown in figure D.13, and fill in the rest of the argument details. An example of a filled-in argument is shown in figure D.14.

Once satisfied, click "Submit" to create the argument. The argument will promptly be shown on its argument page, shown in figure D.15.

### D.2.4 Responding to an Argument

To create an attacking or supporting argument, navigate to an argument's page, shown in figure D.15, and click either "Attack This Argument" or "Support This Argument".

Once a selection has been made, the argument creation page will be shown. When creating a responding argument, critical questions will be shown to help guide the creation of a responding argument. An example of this is shown in figure D.16.

To insert the value from a proposition in the argument which is being attacked or supported, press the plus button on the right-hand side of the field as shown in figure D.17.

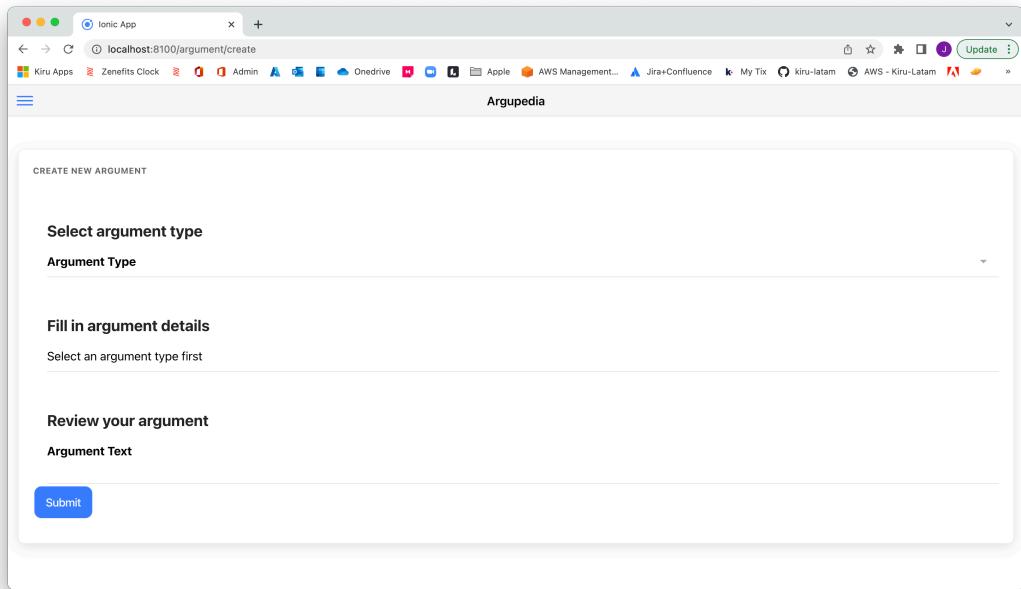


Figure D.12: Argupedia Create Argument Page

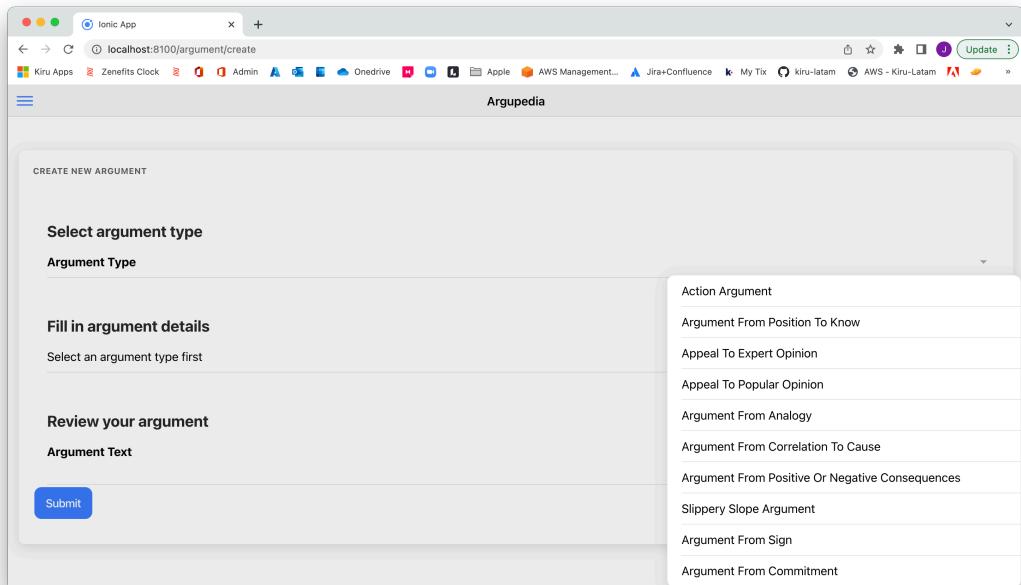


Figure D.13: Argupedia Select Argument Type Dropdown

**Select argument type**

Argument Type: Appeal To Expert Opinion

**Fill in argument details**

Expert: Dr. Klausner

Domain: infectious diseases

Fact: there is a pandemic

Proposition: true

**Review your argument**

Argument Text: Dr. Klausner is an expert in infectious diseases and suggests that it is true that there is a pandemic.

**Submit**

Figure D.14: Argupedia Filled-in Create Argument Page

**SELECTED ARGUMENT**

**Dr. Klausner is an expert in infectious diseases and suggests that it is true that there is a pandemic.**

Argument Type	Appeal To Expert Opinion
Expert	Dr. Klausner
Fact	There is a pandemic
Domain	Infectious diseases
Proposition	True
Author	Jonathan Damico
Last Modified	3 seconds ago

**Edit This Argument** | **Attack This Argument** | **Support This Argument**

Figure D.15: Argupedia Argument Page

**ATTACKING ARGUMENT**

**Dr. Klausner is an expert in infectious diseases and suggests that it is true that there is a pandemic.**

Argument Type	Appeal To Expert Opinion
expert	Dr. Klausner
fact	there is a pandemic
domain	infectious diseases
proposition	true
Last Modified	13 seconds ago

**CRITICAL QUESTIONS**

Use the following critical questions to consider how you would like to respond to the argument:

- How credible is Dr. Klausner as an expert?
- Is Dr. Klausner an expert in the field that there is a pandemic in?
- What did Dr. Klausner assert that implies there is a pandemic?
- Is Dr. Klausner personally reliable and trustworthy? Do we have any reason to think Dr. Klausner is less than honest?
- Is there a pandemic consistent with what other experts assert?
- Is Dr. Klausner's evidence based on evidence?

Figure D.16: Argupedia Create Attacking Argument Page

Once satisfied with the argument created, click the "Submit" button. The new argument will be displayed as shown in figure D.18.

### D.2.5 Editing an Argument

After creating an argument, if the argument text does not necessarily represent the argument well, it is possible to edit this argument text.

From the argument page, as shown in figure D.15, click on "Edit This Argument".

The edit argument page will then be displayed, as shown in figure D.19.

On the edit argument page, edit the text of the argument to what is desired as shown in figure D.20 and click "Save".

The edited argument will then be shown, as demonstrated in figure D.21.

### D.2.6 Navigating an Argument Graph

On an argument page, such as the one in D.18, it is possible to navigate between parts of the argument by clicking nodes in the argument graph on the left-hand side of the page.

Red nodes are labelled out and green nodes are labelled in. Red edges denote an attacking relation and green edges denote a supporting relation.

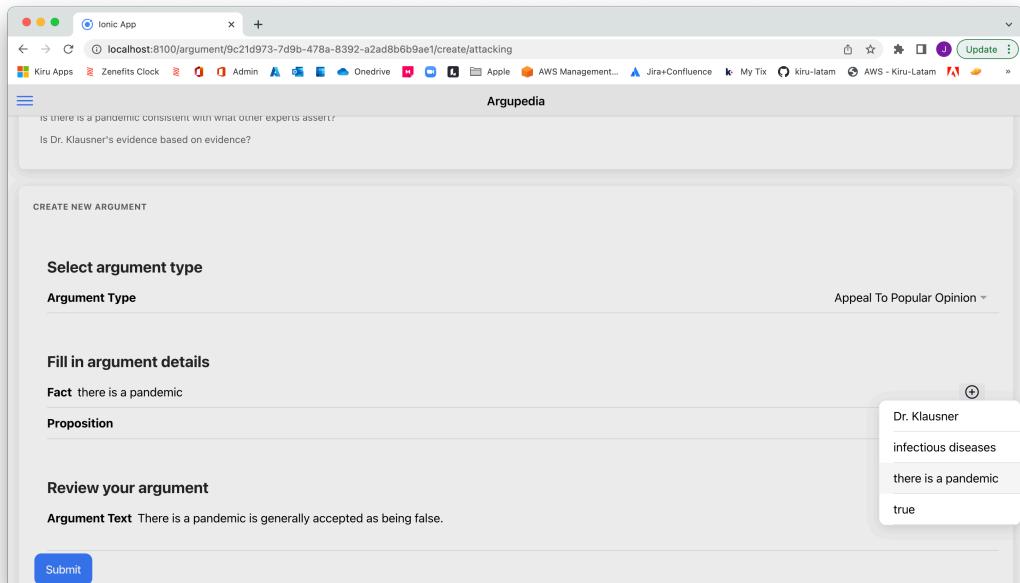


Figure D.17: Argupedia Proposition Insertion

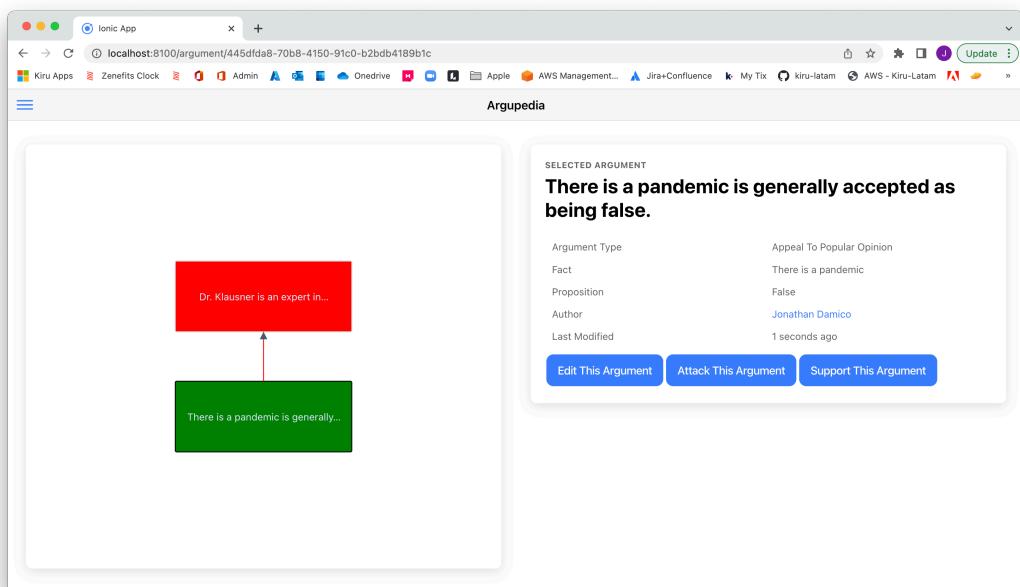


Figure D.18: Argupedia Attacking Argument

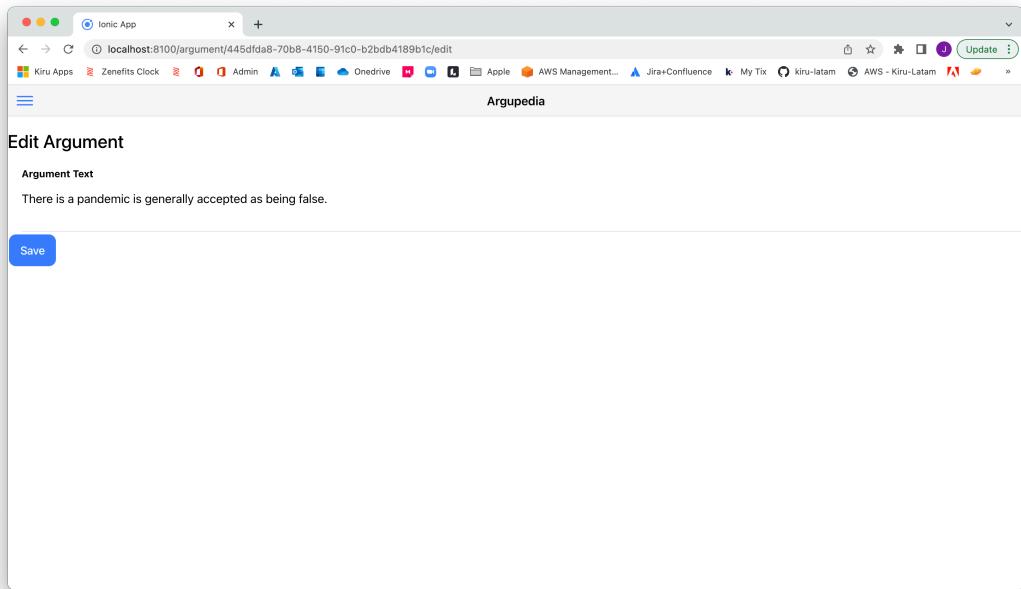


Figure D.19: Argupedia Edit Argument Page

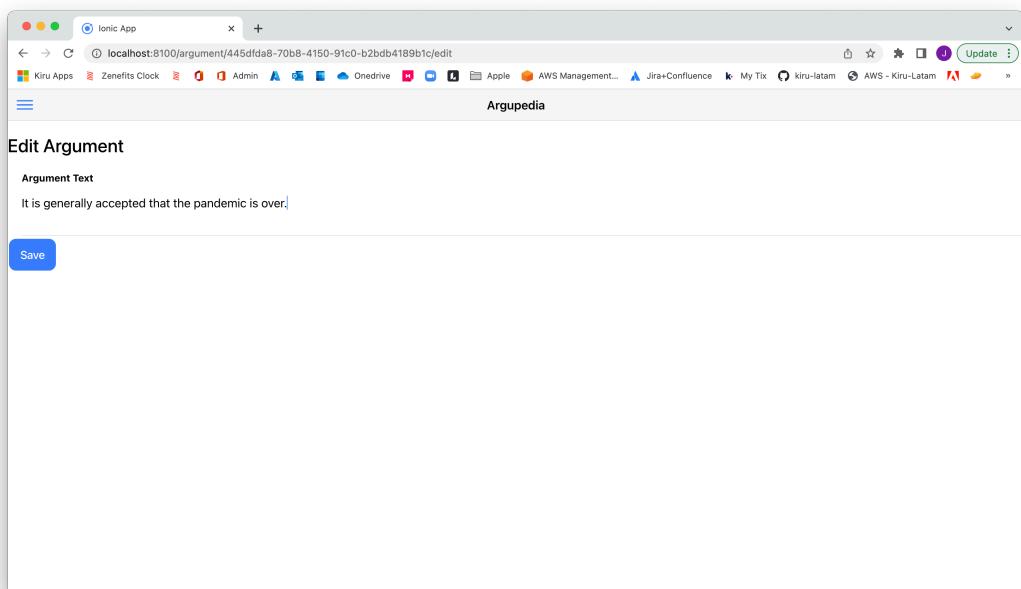


Figure D.20: Argupedia Filled-in Edit Argument Page

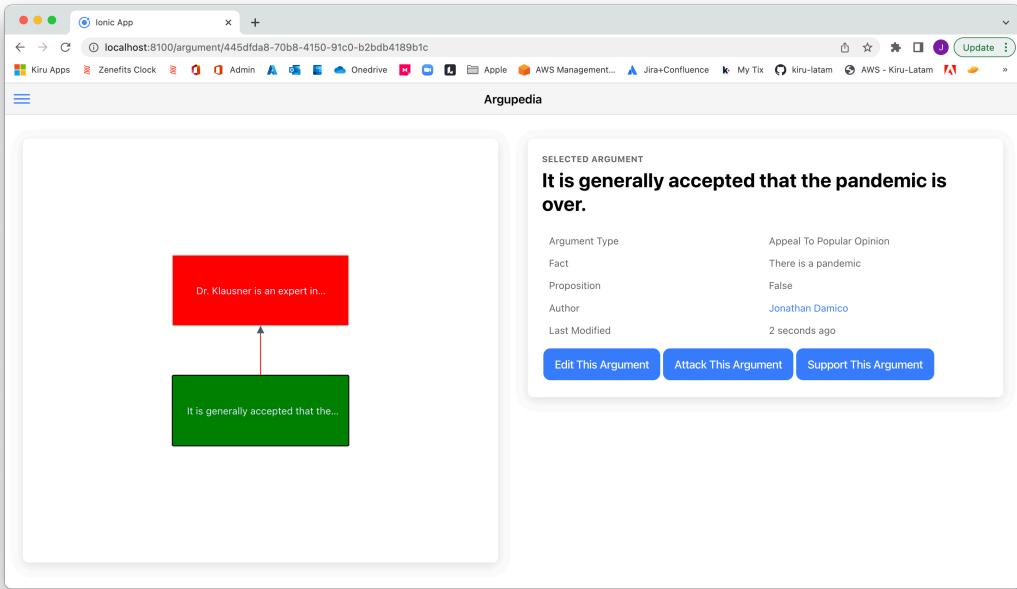


Figure D.21: Argupedia Argument Page After Edits

### D.2.7 Creating a Comment

Lastly, to create a comment on an article, navigate to an argument page and scroll towards the bottom.

Enter a comment in the comment form, as shown in figure D.22 and click "Post" when satisfied.

The comment will then be added to the argument as shown in figure D.23. The argument can be deleted by selecting the "Delete" button.

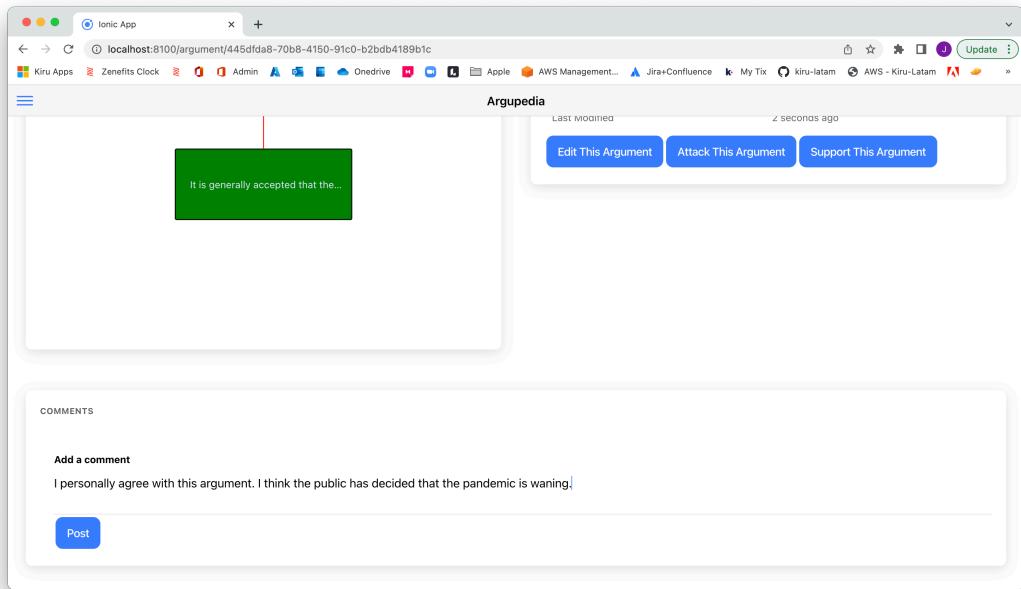


Figure D.22: Argupedia Comment Creation

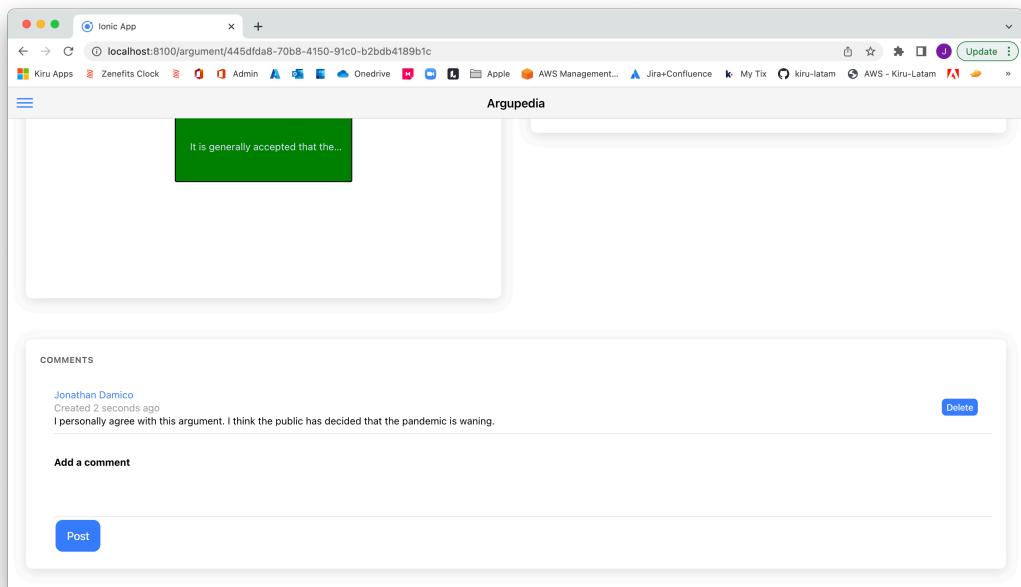


Figure D.23: Argupedia Argument Page With Comment

# Appendix E

## Source Code Listings

I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary.

Jonathan Damico  
April 8, 2022

### Contents

---

<b>E.1 Argupedia API Source Code . . . . .</b>	<b>85</b>
E.1.1 argupedia-api-master/jest-dynamodb-config.js . . . . .	85
E.1.2 argupedia-api-master/jest.config.js . . . . .	88
E.1.3 argupedia-api-master/package.json . . . . .	89
E.1.4 argupedia-api-master/serverless.ts . . . . .	91
E.1.5 argupedia-api-master/sonar-project.properties . . . . .	97
E.1.6 argupedia-api-master/src/adapters/database/argumentTable.test.ts	98
E.1.7 argupedia-api-master/src/adapters/database/argumentTable.ts . . .	104
E.1.8 argupedia-api-master/src/adapters/database/commentTable.test.ts .	111
E.1.9 argupedia-api-master/src/adapters/database/commentTable.ts . . .	113
E.1.10 argupedia-api-master/src/adapters/database/index.ts . . . . .	115
E.1.11 argupedia-api-master/src/adapters/database/userTable.test.ts . . . .	116
E.1.12 argupedia-api-master/src/adapters/database/userTable.ts . . . . .	122
E.1.13 argupedia-api-master/src/functions/clearvoteArgumentWithId/handler.spec.ts	129
E.1.14 argupedia-api-master/src/functions/clearvoteArgumentWithId/handler.ts	133
E.1.15 argupedia-api-master/src/functions/clearvoteArgumentWithId/index.ts	135

E.1.16	argupedia-api-master/src/functions/clearvoteArgumentWithId/mock.json	136
E.1.17	argupedia-api-master/src/functions/clearvoteArgumentWithId/schema.ts	137
E.1.18	argupedia-api-master/src/functions/createCommentForArgumentWithId/handler.spec.ts	138
E.1.19	argupedia-api-master/src/functions/createCommentForArgumentWithId/handler.ts	141
E.1.20	argupedia-api-master/src/functions/createCommentForArgumentWithId/index.ts	143
E.1.21	argupedia-api-master/src/functions/createCommentForArgumentWithId/mock.json	144
E.1.22	argupedia-api-master/src/functions/createCommentForArgumentWithId/schema.ts	145
E.1.23	argupedia-api-master/src/functions/createNewArgument/handler.spec.ts	146
E.1.24	argupedia-api-master/src/functions/createNewArgument/handler.ts	149
E.1.25	argupedia-api-master/src/functions/createNewArgument/index.ts	. 151
E.1.26	argupedia-api-master/src/functions/createNewArgument/mock.json	152
E.1.27	argupedia-api-master/src/functions/createNewArgument/schema.ts	153
E.1.28	argupedia-api-master/src/functions/createNewAttackingArgument/handler.spec.ts	154
E.1.29	argupedia-api-master/src/functions/createNewAttackingArgument/handler.ts	162
E.1.30	argupedia-api-master/src/functions/createNewAttackingArgument/index.ts	165
E.1.31	argupedia-api-master/src/functions/createNewAttackingArgument/mock.json	166
E.1.32	argupedia-api-master/src/functions/createNewAttackingArgument/schema.ts	167
E.1.33	argupedia-api-master/src/functions/createNewSupportingArgument/handler.spec.ts	168
E.1.34	argupedia-api-master/src/functions/createNewSupportingArgument/handler.ts	173
E.1.35	argupedia-api-master/src/functions/createNewSupportingArgument/index.ts	176
E.1.36	argupedia-api-master/src/functions/createNewSupportingArgument/mock.json	177
E.1.37	argupedia-api-master/src/functions/createNewSupportingArgument/schema.ts	178
E.1.38	argupedia-api-master/src/functions/createUser/handler.spec.ts . . .	179
E.1.39	argupedia-api-master/src/functions/createUser/handler.ts . . . . .	181
E.1.40	argupedia-api-master/src/functions/createUser/index.ts . . . . .	183
E.1.41	argupedia-api-master/src/functions/createUser/mock.json . . . . .	185
E.1.42	argupedia-api-master/src/functions/createUser/schema.ts . . . . .	186
E.1.43	argupedia-api-master/src/functions/deleteArgumentWithId/handler.spec.ts	187
E.1.44	argupedia-api-master/src/functions/deleteArgumentWithId/handler.ts	191
E.1.45	argupedia-api-master/src/functions/deleteArgumentWithId/index.ts	193
E.1.46	argupedia-api-master/src/functions/deleteArgumentWithId/mock.json	194
E.1.47	argupedia-api-master/src/functions/deleteArgumentWithId/schema.ts	195
E.1.48	argupedia-api-master/src/functions/deleteCommentWithId/handler.spec.ts	196

E.1.49	argupedia-api-master/src/functions/deleteCommentWithId/handler.ts	199
E.1.50	argupedia-api-master/src/functions/deleteCommentWithId/index.ts	201
E.1.51	argupedia-api-master/src/functions/deleteCommentWithId/mock.json	202
E.1.52	argupedia-api-master/src/functions/deleteCommentWithId/schema.ts	203
E.1.53	argupedia-api-master/src/functions/deleteUserWithId/handler.spec.ts	204
E.1.54	argupedia-api-master/src/functions/deleteUserWithId/handler.ts . .	208
E.1.55	argupedia-api-master/src/functions/deleteUserWithId/index.ts . . .	210
E.1.56	argupedia-api-master/src/functions/deleteUserWithId/mock.json . .	211
E.1.57	argupedia-api-master/src/functions/deleteUserWithId/schema.ts . .	212
E.1.58	argupedia-api-master/src/functions/downvoteArgumentWithId/handler.spec.ts	213
E.1.59	argupedia-api-master/src/functions/downvoteArgumentWithId/handler.ts	218
E.1.60	argupedia-api-master/src/functions/downvoteArgumentWithId/index.ts	220
E.1.61	argupedia-api-master/src/functions/downvoteArgumentWithId/mock.json	221
E.1.62	argupedia-api-master/src/functions/downvoteArgumentWithId/schema.ts	222
E.1.63	argupedia-api-master/src/functions/getAllArguments/handler.spec.ts	223
E.1.64	argupedia-api-master/src/functions/getAllArguments/handler.ts . .	225
E.1.65	argupedia-api-master/src/functions/getAllArguments/index.ts . . .	226
E.1.66	argupedia-api-master/src/functions/getAllArguments/mock.json . .	227
E.1.67	argupedia-api-master/src/functions/getAllArguments/schema.ts . .	228
E.1.68	argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/handler.spec.ts	229
E.1.69	argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/handler.ts	232
E.1.70	argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/index.ts	234
E.1.71	argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/mock.json	235
E.1.72	argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/schema.ts	236
E.1.73	argupedia-api-master/src/functions/getArgumentWithId/handler.spec.ts	237
E.1.74	argupedia-api-master/src/functions/getArgumentWithId/handler.ts	239
E.1.75	argupedia-api-master/src/functions/getArgumentWithId/index.ts .	240
E.1.76	argupedia-api-master/src/functions/getArgumentWithId/mock.json	241
E.1.77	argupedia-api-master/src/functions/getArgumentWithId/schema.ts	242
E.1.78	argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/handler.spec.ts	243
E.1.79	argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/handler.ts	246
E.1.80	argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/index.ts	248
E.1.81	argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/mock.json	249

E.1.82	argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/schema.ts	250
E.1.83	argupedia-api-master/src/functions/getAuthoredCommentsForUserWithId/handler.spec.ts	251
E.1.84	argupedia-api-master/src/functions/getAuthoredCommentsForUserWithId/handler.ts	254
E.1.85	argupedia-api-master/src/functions/getAuthoredCommentsForUserWithId/index.ts	256
E.1.86	argupedia-api-master/src/functions/getAuthoredCommentsForUserWithId/mock.json	257
E.1.87	argupedia-api-master/src/functions/getAuthoredCommentsForUserWithId/schema.ts	258
E.1.88	argupedia-api-master/src/functions/getCommentWithId/handler.spec.ts	259
E.1.89	argupedia-api-master/src/functions/getCommentWithId/handler.ts	261
E.1.90	argupedia-api-master/src/functions/getCommentWithId/index.ts . .	262
E.1.91	argupedia-api-master/src/functions/getCommentWithId/mock.json	263
E.1.92	argupedia-api-master/src/functions/getCommentWithId/schema.ts .	264
E.1.93	argupedia-api-master/src/functions/getCommentsForArgumentWithId/handler.spec.ts	265
E.1.94	argupedia-api-master/src/functions/getCommentsForArgumentWithId/handler.ts	268
E.1.95	argupedia-api-master/src/functions/getCommentsForArgumentWithId/index.ts	270
E.1.96	argupedia-api-master/src/functions/getCommentsForArgumentWithId/mock.json	271
E.1.97	argupedia-api-master/src/functions/getCommentsForArgumentWithId/schema.ts	272
E.1.98	argupedia-api-master/src/functions/getUserWithId/handler.spec.ts .	273
E.1.99	argupedia-api-master/src/functions/getUserWithId/handler.ts . . .	275
E.1.100	argupedia-api-master/src/functions/getUserWithId/index.ts . . . .	276
E.1.101	argupedia-api-master/src/functions/getUserWithId/mock.json . . .	277
E.1.102	argupedia-api-master/src/functions/getUserWithId/schema.ts . . . .	278
E.1.103	argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/handler.spec.ts	279
E.1.104	argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/handler.ts	282
E.1.105	argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/index.ts	284
E.1.106	argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/mock.json	285
E.1.107	argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/schema.ts	286
E.1.108	argupedia-api-master/src/functions/index.ts . . . . . . . . . .	287
E.1.109	argupedia-api-master/src/functions/updateArgumentWithId/handler.spec.ts	288
E.1.110	argupedia-api-master/src/functions/updateArgumentWithId/handler.ts	292
E.1.111	argupedia-api-master/src/functions/updateArgumentWithId/index.ts	295
E.1.112	argupedia-api-master/src/functions/updateArgumentWithId/mock.json	296
E.1.113	argupedia-api-master/src/functions/updateArgumentWithId/schema.ts	297
E.1.114	argupedia-api-master/src/functions/updateCommentWithId/handler.spec.ts	298

E.1.115argupedia-api-master/src/functions/updateCommentWithId/handler.ts	301
E.1.116argupedia-api-master/src/functions/updateCommentWithId/index.ts	303
E.1.117argupedia-api-master/src/functions/updateCommentWithId/mock.json	304
E.1.118argupedia-api-master/src/functions/updateCommentWithId/schema.ts	305
E.1.119argupedia-api-master/src/functions/updateUserWithId/handler.spec.ts	306
E.1.120argupedia-api-master/src/functions/updateUserWithId/handler.ts	309
E.1.121argupedia-api-master/src/functions/updateUserWithId/index.ts	311
E.1.122argupedia-api-master/src/functions/updateUserWithId/mock.json	312
E.1.123argupedia-api-master/src/functions/updateUserWithId/schema.ts	313
E.1.124argupedia-api-master/src/functions/upvoteArgumentWithId/handler.spec.ts	314
E.1.125argupedia-api-master/src/functions/upvoteArgumentWithId/handler.ts	319
E.1.126argupedia-api-master/src/functions/upvoteArgumentWithId/index.ts	321
E.1.127argupedia-api-master/src/functions/upvoteArgumentWithId/mock.json	322
E.1.128argupedia-api-master/src/functions/upvoteArgumentWithId/schema.ts	323
E.1.129argupedia-api-master/src/helpers/Cognito/index.ts	324
E.1.130argupedia-api-master/src/helpers/DynamoDB/config.ts	325
E.1.131argupedia-api-master/src/helpers/DynamoDB/index.ts	326
E.1.132argupedia-api-master/src/helpers/JWT/index.test.ts	327
E.1.133argupedia-api-master/src/helpers/JWT/index.ts	328
E.1.134argupedia-api-master/src/helpers/argupedia/index.ts	329
E.1.135argupedia-api-master/src/helpers/argupedia/labelArguments.test.ts	330
E.1.136argupedia-api-master/src/helpers/argupedia/labelArguments.ts	335
E.1.137argupedia-api-master/src/helpers/argupedia/validateArgument.ts	337
E.1.138argupedia-api-master/src/helpers/faker/argument.ts	341
E.1.139argupedia-api-master/src/helpers/faker/comment.ts	350
E.1.140argupedia-api-master/src/helpers/faker/index.ts	351
E.1.141argupedia-api-master/src/helpers/faker/user.ts	352
E.1.142argupedia-api-master/src/helpers/mergeOptions/index.ts	353
E.1.143argupedia-api-master/src/helpers/mergeOptions/mergeOptions.ts	354
E.1.144argupedia-api-master/src/libs/apiGateway.ts	355
E.1.145argupedia-api-master/src/libs/handlerResolver.ts	357
E.1.146argupedia-api-master/src/libs/lambda.ts	358
E.1.147argupedia-api-master/src/types/arguments.ts	359

E.1.148	argupedia-api-master/src/types/comments.ts . . . . .	363
E.1.149	argupedia-api-master/src/types/recursivePartial.ts . . . . .	364
E.1.150	argupedia-api-master/src/types/users.ts . . . . .	365
E.1.151	argupedia-api-master/tsconfig.json . . . . .	366
E.1.152	argupedia-api-master/tsconfig.paths.json . . . . .	367
<b>E.2</b>	<b>Argupedia App Source Code . . . . .</b>	<b>368</b>
E.2.1	argupedia-app-master/capacitor.config.ts . . . . .	368
E.2.2	argupedia-app-master/ionic.config.json . . . . .	369
E.2.3	argupedia-app-master/jest.config.js . . . . .	370
E.2.4	argupedia-app-master/package.json . . . . .	371
E.2.5	argupedia-app-master/public/manifest.json . . . . .	374
E.2.6	argupedia-app-master/sonar-project.properties . . . . .	375
E.2.7	argupedia-app-master/src/App.test.tsx . . . . .	376
E.2.8	argupedia-app-master/src/App.tsx . . . . .	377
E.2.9	argupedia-app-master/src/aws-exports.js . . . . .	381
E.2.10	argupedia-app-master/src/components/ArgumentCard.css . . . . .	382
E.2.11	argupedia-app-master/src/components/ArgumentCard.tsx . . . . .	383
E.2.12	argupedia-app-master/src/components/Box/Box.tsx . . . . .	384
E.2.13	argupedia-app-master/src/components/Box/index.ts . . . . .	385
E.2.14	argupedia-app-master/src/components/CommentForm.tsx . . . . .	386
E.2.15	argupedia-app-master/src/components/Toolbar.tsx . . . . .	388
E.2.16	argupedia-app-master/src/constants/argumentTypePlaceholders.ts . . . . .	390
E.2.17	argupedia-app-master/src/constants/argumentTypeSchema.ts . . . . .	393
E.2.18	argupedia-app-master/src/constants/criticalQuestions.ts . . . . .	396
E.2.19	argupedia-app-master/src/helpers/argupedia/argumentTranslations.ts . . . . .	399
E.2.20	argupedia-app-master/src/helpers/argupedia/capitalizeFirstLetter.ts . . . . .	404
E.2.21	argupedia-app-master/src/helpers/argupedia/formatCriticalQuestions.ts . . . . .	405
E.2.22	argupedia-app-master/src/helpers/argupedia/pascalCaseToSentence.ts . . . . .	406
E.2.23	argupedia-app-master/src/helpers/argupedia/timeDifference.ts . . . . .	407
E.2.24	argupedia-app-master/src/helpers/mergeOptions/index.ts . . . . .	408
E.2.25	argupedia-app-master/src/helpers/mergeOptions/mergeOptions.ts . . . . .	409
E.2.26	argupedia-app-master/src/hooks/index.ts . . . . .	410
E.2.27	argupedia-app-master/src/hooks/useStore.ts . . . . .	411

E.2.28 argupedia-app-master/src/index.tsx . . . . .	412
E.2.29 argupedia-app-master/src/pages/Argument.css . . . . .	413
E.2.30 argupedia-app-master/src/pages/Argument.tsx . . . . .	414
E.2.31 argupedia-app-master/src/pages/CreateArgument.css . . . . .	425
E.2.32 argupedia-app-master/src/pages/CreateArgument.tsx . . . . .	426
E.2.33 argupedia-app-master/src/pages/EditArgument.css . . . . .	438
E.2.34 argupedia-app-master/src/pages/EditArgument.tsx . . . . .	439
E.2.35 argupedia-app-master/src/pages/EditProfile.css . . . . .	442
E.2.36 argupedia-app-master/src/pages/EditProfile.tsx . . . . .	443
E.2.37 argupedia-app-master/src/pages/Home.css . . . . .	447
E.2.38 argupedia-app-master/src/pages/Home.tsx . . . . .	448
E.2.39 argupedia-app-master/src/pages/Profile.css . . . . .	450
E.2.40 argupedia-app-master/src/pages/Profile.tsx . . . . .	451
E.2.41 argupedia-app-master/src/react-app-env.d.ts . . . . .	456
E.2.42 argupedia-app-master/src/reportWebVitals.ts . . . . .	457
E.2.43 argupedia-app-master/src/service-worker.ts . . . . .	458
E.2.44 argupedia-app-master/src/serviceWorkerRegistration.ts . . . . .	460
E.2.45 argupedia-app-master/src/services/api/api.ts . . . . .	464
E.2.46 argupedia-app-master/src/services/api/index.ts . . . . .	473
E.2.47 argupedia-app-master/src/services/index.ts . . . . .	474
E.2.48 argupedia-app-master/src/setupTests.ts . . . . .	475
E.2.49 argupedia-app-master/src/theme/variables.css . . . . .	476
E.2.50 argupedia-app-master/src/types/arguments.ts . . . . .	482
E.2.51 argupedia-app-master/src/types/comments.ts . . . . .	487
E.2.52 argupedia-app-master/src/types/index.ts . . . . .	488
E.2.53 argupedia-app-master/src/types/recursivePartial.ts . . . . .	489
E.2.54 argupedia-app-master/src/types/users.ts . . . . .	490
E.2.55 argupedia-app-master/tsconfig.json . . . . .	491

---

Files containing "serverless", "config", "package", "manifest", and "properties" are not entirely authored by Jonathan Damico and may contain boilerplate code from the organisations and individuals responsible for them.

## E.1 Argupedia API Source Code

### E.1.1 argupedia-api-master/jest-dynamodb-config.js

```
module.exports = {

  tables: [
    {
      TableName: process.env.ARGUMENT_TABLE_NAME,
      AttributeDefinitions: [
        {
          AttributeName: 'argumentId',
          AttributeType: 'S',
        },
      ],
      KeySchema: [
        {
          AttributeName: 'argumentId',
          KeyType: 'HASH',
        },
      ],
      ProvisionedThroughput: {
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1,
      },
    },
    {
      TableName: process.env.COMMENT_TABLE_NAME,
      AttributeDefinitions: [
        {
          AttributeName: 'commentId',
          AttributeType: 'S',
        },
      ],
      KeySchema: [
        {
          AttributeName: 'commentId',
          KeyType: 'HASH',
        },
      ],
      ProvisionedThroughput: {
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1,
      }
    }
  ]
}
```

```

    },
    },
    {
      TableName: process.env.USER_TABLE_NAME,
      AttributeDefinitions: [
        {
          AttributeName: 'userId',
          AttributeType: 'S',
        },
      ],
      KeySchema: [
        {
          AttributeName: 'userId',
          KeyType: 'HASH',
        },
      ],
      ProvisionedThroughput: {
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1,
      },
    },
  ],
};

// module.exports = async () => {
//   const serverless = new (require('serverless'))();
//   // console.log(serverless);
//   console.log(serverless.config);
//   // If using monorepo where DynamoDB serverless.yml is in another directory
//   const serverless = new (require('serverless'))({ servicePath: 'serverless.ts' });
//   await serverless.init();
//   const service = await serverless.variables.populateService();
//   const resources = service.resources.filter(r => Object.keys(r).includes('Resources'))[0];

//   const tables = Object.keys(resources)
//     .map(name => resources[name])
//     .filter(r => r.Type === 'AWS::DynamoDB::Table')
//     .map(r => r.Properties);

//   return {
//     tables
//   };
}

```



### E.1.2 argupedia-api-master/jest.config.js

```
const tsconfig = require('./tsconfig.paths.json');
const moduleNameMapper = require('tsconfig-paths-jest')(tsconfig);

require('dotenv').config();

module.exports = {
  roots: ['<rootDir>/src'],
  testMatch: [
    '**/_tests_/**/*.(ts|tsx|js)',
    '**/?(*.)+(spec|test).+(ts|tsx|js)',
  ],
  transform: {
    '^.+\\.(ts|tsx)$': 'ts-jest',
  },
  moduleFileExtensions: ['ts', 'tsx', 'js', 'jsx', 'json', 'node'],
  preset: '@shelf/jest-dynamodb',
  moduleNameMapper,
};

};
```

### E.1.3 argupedia-api-master/package.json

```
{  
  "name": "argupedia-api",  
  "version": "1.0.0",  
  "description": "Serverless\u00a0aws-nodejs-typescript\u00a0template",  
  "main": "serverless.ts",  
  "scripts": {  
    "export-env": "sls\u00a0export-env",  
    "build": "sls\u00a0package",  
    "test": "jest",  
    "format": "prettier\u00a0--loglevel\u00a0warn\u00a0--write\u00a0\"**/*.{ts,js,json}\\"",  
    "format:check": "prettier\u00a0--loglevel\u00a0warn\u00a0--check\u00a0\"**/*.{ts,js,json}\\"",  
    "lint": "eslint\u00a0\"**/*.ts\"\u00a0--fix",  
    "lint:check": "eslint\u00a0\"**/*.ts\\""  
  },  
  "engines": {  
    "node": ">=14.15.0"  
  },  
  "dependencies": {  
    "@aws-sdk/client-cognito-identity-provider": "3.58.0",  
    "@midway/core": "^2.5.3",  
    "@midway/http-json-body-parser": "^2.5.3",  
    "@types/uuid": "8.3.4",  
    "aws-jwt-verify": "2.1.3",  
    "uuid": "8.3.2"  
  },  
  "devDependencies": {  
    "@faker-js/faker": "6.0.0-alpha.7",  
    "@serverless/typescript": "^2.23.0",  
    "@shelf/jest-dynamodb": "2.2.3",  
    "@types/aws-lambda": "8.10.71",  
    "@types/jest": "27.4.1",  
    "@types/node": "^14.14.25",  
    "@typescript-eslint/eslint-plugin": "5.5.0",  
    "@typescript-eslint/parser": "5.5.0",  
    "aws-lambda-mock-context": "3.2.1",  
    "deepmerge": "4.2.2",  
    "eslint": "8.3.0",  
    "eslint-config-prettier": "8.3.0",  
    "eslint-plugin-import": "2.25.3",  
    "eslint-plugin-security": "1.4.0",  
  }  
}
```

```
"eslint-plugin-simple-import-sort": "7.0.0",
"jest": "27.5.1",
"json-schema-to-ts": "^1.5.0",
"prettier": "2.5.0",
"serverless": "^2.23.0",
"serverless-dotenv-plugin": "3.10.0",
"serverless-esbuild": "^1.17.1",
"serverless-export-env": "github:arabold/serverless-export-env",
"serverless-jest-plugin": "0.4.0",
"ts-jest": "27.1.3",
"ts-node": "^10.4.0",
"tsconfig-paths": "^3.9.0",
"tsconfig-paths-jest": "0.0.1",
"typescript": "^4.1.3"
},
"author": "The\u2014serverless\u2014webpack\u2014authors\u2014(https://github.com/elastic-coders/serverless-webpack)",
"license": "MIT",
"jestSonar": {
  "withRelativePaths": true
}
}
```

#### E.1.4 argupedia-api-master/serverless.ts

```
import clearvoteArgumentWithId from '@functions/clearvoteArgumentWithId';
import createCommentForArgumentWithId from '@functions/createCommentForArgumentWithId';
import createNewArgument from '@functions/createNewArgument';
import createNewAttackingArgument from '@functions/createNewAttackingArgument';
import createNewSupportingArgument from '@functions/createNewSupportingArgument';
import createUser from '@functions/createUser';
import deleteArgumentWithId from '@functions/deleteArgumentWithId';
import deleteCommentWithId from '@functions/deleteCommentWithId';
import deleteUserWithId from '@functions/deleteUserWithId';
import downvoteArgumentWithId from '@functions/downvoteArgumentWithId';
import getAllArguments from '@functions/getAllArguments';
import getArgumentGraphForArgumentWithId from '@functions/getArgumentGraphForArgumentWithId';
import getArgumentWithId from '@functions/getArgumentWithId';
import getAuthoredArgumentsForUserWithId from '@functions/getAuthoredArgumentsForUserWithId';
import getAuthoredCommentsForUserWithId from '@functions/getAuthoredCommentsForUserWithId';
import getCommentsForArgumentWithId from '@functions/getCommentsForArgumentWithId';
import getCommentWithId from '@functions/getCommentWithId';
import getUserWithId from '@functions/getUserWithId';
import getVotedArgumentsForUserWithId from '@functions/getVotedArgumentsForUserWithId';
import updateArgumentWithId from '@functions/updateArgumentWithId';
import updateCommentWithId from '@functions/updateCommentWithId';
import updateUserWithId from '@functions/updateUserWithId';
import upvoteArgumentWithId from '@functions/upvoteArgumentWithId';
import type { AWS } from '@serverless/typescript';

const serverlessConfiguration: AWS = {
  service: 'argupedia-api',
  frameworkVersion: '2',
  plugins: [
    'serverless-esbuild',
    'serverless-jest-plugin',
    'serverless-export-env',
  ],
  provider: {
    name: 'aws',
    runtime: 'nodejs14.x',
    region: 'eu-west-2',
    apiGateway: {
      minimumCompressionSize: 1024,
      shouldStartNameWithService: true,
    }
  }
}
```

```

},
environment: {
  AWS_NODEJS_CONNECTION_REUSE_ENABLED: '1',
  NODE_OPTIONS: '--enable-source-maps--stack-trace-limit=1000',
  USER_TABLE_NAME:
    '${self:service}-${opt:stage,${self:provider.stage}}-UserTable',
  ARGUMENT_TABLE_NAME:
    '${self:service}-${opt:stage,${self:provider.stage}}-ArgumentTable',
  COMMENT_TABLE_NAME:
    '${self:service}-${opt:stage,${self:provider.stage}}-CommentTable',
  DISABLE_MOCKED_WARNING: 'true',
  COGNITO_USER_POOL_ID: {
    Ref: 'CognitoUserPool',
  },
  COGNITO_APP_CLIENT_ID: {
    Ref: 'CognitoUserPoolClient',
  },
},
lambdaHashingVersion: '20201221',
iam: {
  role: {
    statements: [
      {
        Effect: 'Allow',
        Action: ['dynamodb:*'],
        Resource: ['*'],
      },
      {
        Effect: 'Allow',
        Action: ['cognito-idp:AdminDeleteUser'],
        Resource: ['*'],
      },
    ],
  },
},
functions: {
  createNewArgument,
  createNewAttackingArgument,
  createNewSupportingArgument,
  getArgumentWithId,
  getAllArguments,
}

```

```

deleteArgumentWithId,
upvoteArgumentWithId,
downvoteArgumentWithId,
clearvoteArgumentWithId,
createCommentForArgumentWithId,
getCommentsForArgumentWithId,
getCommentWithId,
deleteCommentWithId,
updateCommentWithId,
updateUserWithId,
deleteUserWithId,
getUserWithId,
createUser,
updateArgumentWithId,
getAuthoredArgumentsForUserWithId,
getAuthoredCommentsForUserWithId,
getVotedArgumentsForUserWithId,
getArgumentGraphForArgumentWithId,
},
resources: {
  Resources: {
    UserTable: {
      Type: 'AWS::DynamoDB::Table',
      DeletionPolicy: 'Delete',
      Properties: {
        AttributeDefinitions: [
          {
            AttributeName: 'userId',
            AttributeType: 'S',
          },
        ],
        KeySchema: [
          {
            AttributeName: 'userId',
            KeyType: 'HASH',
          },
        ],
        ProvisionedThroughput: {
          ReadCapacityUnits: 1,
          WriteCapacityUnits: 1,
        },
      },
      TableName: '${self:provider.environment.USER_TABLE_NAME}',
    }
  }
}

```

```

},
},
ArgumentTable: {
  Type: 'AWS::DynamoDB::Table',
  DeletionPolicy: 'Delete',
  Properties: {
    AttributeDefinitions: [
      {
        AttributeName: 'argumentId',
        AttributeType: 'S',
      },
    ],
    KeySchema: [
      {
        AttributeName: 'argumentId',
        KeyType: 'HASH',
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
    TableName: '${self:provider.environment.ARGUMENT_TABLE_NAME}',
  },
},
CommentTable: {
  Type: 'AWS::DynamoDB::Table',
  DeletionPolicy: 'Delete',
  Properties: {
    AttributeDefinitions: [
      {
        AttributeName: 'commentId',
        AttributeType: 'S',
      },
    ],
    KeySchema: [
      {
        AttributeName: 'commentId',
        KeyType: 'HASH',
      },
    ],
    ProvisionedThroughput: {

```

```

    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: '${self:provider.environment.COMMENT_TABLE_NAME}',
},
},
CognitoUserPool: {
  Type: 'AWS::Cognito::UserPool',
  DeletionPolicy: 'Delete',
  Properties: {
    UserPoolName:
      '${self:service}-${opt:stage,${self:provider.stage}}-user-pool',
    UsernameAttributes: ['email'],
    AutoVerifiedAttributes: ['email'],
  },
},
CognitoUserPoolClient: {
  Type: 'AWS::Cognito::UserPoolClient',
  DeletionPolicy: 'Delete',
  Properties: {
    ClientName:
      '${self:service}-${opt:stage,${self:provider.stage}}-user-pool-client',
    UserPoolId: {
      Ref: 'CognitoUserPool',
    },
    ExplicitAuthFlows: ['ADMIN_NO_SRP_AUTH'],
    GenerateSecret: false,
  },
},
ApiGatewayAuthorizer: {
  Type: 'AWS::ApiGateway::Authorizer',
  DependsOn: ['ApiGatewayRestApi'],
  Properties: {
    RestApiId: {
      Ref: 'ApiGatewayRestApi',
    },
    Name: '${self:service}-${opt:stage,${self:provider.stage}}-cognito-authorizer',
    Type: 'COGNITO_USER_POOLS',
    IdentitySource: 'method.request.header.Authorization',
    ProviderARNs: [
    {
      Fn::GetAtt: ['CognitoUserPool', 'Arn'],

```

```

},
],
},
},
},
GatewayResponseDefault4XX: {
  Type: 'AWS::ApiGateway::GatewayResponse',
  Properties: {
    ResponseParameters: {
      'gatewayresponse.header.Access-Control-Allow-Origin': "'*'",
      'gatewayresponse.header.Access-Control-Allow-Headers': "'*'",
    },
    ResponseType: 'DEFAULT_4XX',
    RestApiId: {
      Ref: 'ApiGatewayRestApi',
    },
  },
},
},
package: { individually: true },
custom: {
  esbuild: {
    bundle: true,
    minify: false,
    sourcemap: true,
    exclude: ['aws-sdk'],
    target: 'node14',
    define: { 'require.resolve': undefined },
    platform: 'node',
    concurrency: 10,
  },
  documentation: {
    version: '1',
    title: 'Argupedia API',
    description: 'Backend layer for Argupedia',
    models: {},
  },
},
};

module.exports = serverlessConfiguration;

```

## E.1.5 argupedia-api-master/sonar-project.properties

```
sonar.organization=jononon
sonar.projectKey=jononon_argupedia-api

# relative paths to source directories. More details and properties are described
# in https://sonarcloud.io/documentation/project-administration/narrowing-the-focus/
sonar.sources=src
sonar.exclusions=src/**/*.test.ts, src/**/*.spec.ts

sonar.tests=src
sonar.test.inclusions=src/**/*.test.ts, src/**/*.spec.ts

sonar.javascript.lcov.reportPaths=coverage/lcov.info
```

## E.1.6 argupedia-api-master/src/adapters/database/argumentTable.test.ts

```
import { DynamoDB } from 'aws-sdk';

import { config } from '../../helpers/DynamoDB/config';
import faker from '../../helpers/faker';
import {

  addCommentIdToArgumentCommentIds,
  addUserIdToArgumentDownvotes,
  addUserIdToArgumentUpvotes,
  deleteArgumentById,
  getArgumentById,
  putArgument,
  removeCommentIdFromArgumentCommentIds,
  removeUserIdFromArgumentDownvotes,
  removeUserIdFromArgumentUpvotes,
} from './argumentTable';

const { ARGUMENT_TABLE_NAME } = process.env;

const dynamoDb = new DynamoDB.DocumentClient(config);

it('shouldPutArgumentIntoTable', async () => {
  const argument = faker.argumentTableEntry();

  await putArgument(argument);

  const { Item } = await dynamoDb
    .get({
      TableName: ARGUMENT_TABLE_NAME,
      Key: { argumentId: argument.argumentId },
    })
    .promise();

  expect(Item).toEqual(argument);
});

it('shouldGetArgumentFromTableById', async () => {
  const argument = faker.argumentTableEntry();

  await dynamoDb
    .put({ TableName: ARGUMENT_TABLE_NAME, Item: argument })
```

```

.promise();

const retrievedArgument = await getArgumentById(argument.argumentId);

expect(retrievedArgument).toEqual(argument);
});

it('shouldDeleteArgumentFromTableById', async () => {
  const argument = faker.argumentTableEntry();

  await dynamoDb
    .put({ TableName: ARGUMENT_TABLE_NAME, Item: argument })
    .promise();

  await deleteArgumentById(argument.argumentId);

  const response = await dynamoDb
    .get({
      TableName: ARGUMENT_TABLE_NAME,
      Key: { argumentId: argument.argumentId },
    })
    .promise();

  expect(response).toEqual({});
});

it('shouldUpdateArgumentInTable', async () => {
  const argument = faker.argumentTableEntry();

  await dynamoDb
    .put({ TableName: ARGUMENT_TABLE_NAME, Item: argument })
    .promise();

  const newArgument = faker.argumentTableEntry({
    argumentId: argument.argumentId,
  });

  await putArgument(newArgument);

  const { Item } = await dynamoDb
    .get({
      TableName: ARGUMENT_TABLE_NAME,
    })
    .promise();
});

```

```

    Key: { argumentId: argument.argumentId },
  })
  .promise();

expect(Item).toEqual(newArgument);
});

it('should_add_user_id_to_argument_upvotes', async () => {
  const argument = faker.argumentTableEntry();

  await dynamoDb
    .put({ TableName: ARGUMENT_TABLE_NAME, Item: argument })
    .promise();

  const userIdToAdd = faker.datatype.uuid();

  await addUserIdToArgumentUpvotes(argument.argumentId, userIdToAdd);

  const { Item } = await dynamoDb
    .get({
      TableName: ARGUMENT_TABLE_NAME,
      Key: { argumentId: argument.argumentId },
    })
    .promise();

  expect(Item.upvotedByUserIds).toContain(userIdToAdd);
});

it('should_remove_user_id_from_argument_upvotes', async () => {
  const userIdToBeRemoved = faker.datatype.uuid();

  const argument = faker.argumentTableEntry({
    upvotedByUserIds: [
      ...Array.from({ length: faker.datatype.number(20) }, () =>
        faker.datatype.uuid()
      ),
      userIdToBeRemoved,
    ],
  });

  await dynamoDb
    .put({ TableName: ARGUMENT_TABLE_NAME, Item: argument })

```

```

.promise();

await removeUserIdFromArgumentUpvotes(argument.argumentId, userIdToBeRemoved);

const { Item } = await dynamoDb
.get({
  TableName: ARGUMENT_TABLE_NAME,
  Key: { argumentId: argument.argumentId },
})
.promise();

expect(Item.upvotedByUserIds).not.toContain(userIdToBeRemoved);
});

it('should_add_user_id_to_argument_downvotes', async () => {
  const argument = faker.argumentTableEntry();

  await dynamoDb
    .put({ TableName: ARGUMENT_TABLE_NAME, Item: argument })
    .promise();

  const userIdToAdd = faker.datatype.uuid();

  await addUserIdToArgumentDownvotes(argument.argumentId, userIdToAdd);

  const { Item } = await dynamoDb
    .get({
      TableName: ARGUMENT_TABLE_NAME,
      Key: { argumentId: argument.argumentId },
    })
    .promise();

  expect(Item.downvotedByUserIds).toContain(userIdToAdd);
});

it('should_remove_user_id_from_argument_downvotes', async () => {
  const userIdToBeRemoved = faker.datatype.uuid();

  const argument = faker.argumentTableEntry({
    downvotedByUserIds: [
      ...Array.from({ length: faker.datatype.number(20) }, () =>
        faker.datatype.uuid()
      )
    ],
  });

```

```

),
userIdToBeRemoved,
],
});

await dynamoDb
.put({ TableName: ARGUMENT_TABLE_NAME, Item: argument })
.promise();

await removeUserIdFromArgumentDownvotes(
argument.argumentId,
userIdToBeRemoved
);

const { Item } = await dynamoDb
.get({
TableName: ARGUMENT_TABLE_NAME,
Key: { argumentId: argument.argumentId },
})
.promise();

expect(Item.downvotedByUserIds).not.toContain(userIdToBeRemoved);
});

it('should_add_comment_id_to_argument_comment_ids', async () => {
const argument = faker.argumentTableEntry();

await dynamoDb
.put({ TableName: ARGUMENT_TABLE_NAME, Item: argument })
.promise();

const commentIdToAdd = faker.datatype.uuid();

await addCommentIdToArgumentCommentIds(argument.argumentId, commentIdToAdd);

const { Item } = await dynamoDb
.get({
TableName: ARGUMENT_TABLE_NAME,
Key: { argumentId: argument.argumentId },
})
.promise();

```

```

expect(Item.commentIds).toContain(commentIdToAdd);

});

it('should_remove_comment_id_from_argument_comment_ids', async () => {
  const commentIdToBeRemoved = faker.datatype.uuid();

  const argument = faker.argumentTableEntry({
    commentIds: [
      ...Array.from({ length: faker.datatype.number(20) }, () =>
        faker.datatype.uuid()
      ),
      commentIdToBeRemoved,
    ],
  });
  await dynamoDb
    .put({ TableName: ARGUMENT_TABLE_NAME, Item: argument })
    .promise();

  await removeCommentIdFromArgumentCommentIds(
    argument.argumentId,
    commentIdToBeRemoved
  );
  const { Item } = await dynamoDb
    .get({
      TableName: ARGUMENT_TABLE_NAME,
      Key: { argumentId: argument.argumentId },
    })
    .promise();
  expect(Item.commentIds).not.toContain(commentIdToBeRemoved);
});

```

### E.1.7 argupedia-api-master/src/adapters/database/argumentTable.ts

```
import { DynamoDB } from 'aws-sdk';

import { ArgumentTableEntry } from 'src/types/arguments';

import { config } from '../../helpers/DynamoDB/config';

const dynamoDb = new DynamoDB.DocumentClient(config);

const { ARGUMENT_TABLE_NAME } = process.env;

const putArgument = async (argument: ArgumentTableEntry) => {
  const params: DynamoDB.DocumentClient.PutItemInput = {
    TableName: ARGUMENT_TABLE_NAME,
    Item: argument,
  };

  const result = await dynamoDb.put(params).promise();

  return result;
};

const getArguments = async () => {
  const params: DynamoDB.DocumentClient.ScanInput = {
    TableName: ARGUMENT_TABLE_NAME,
  };

  const result = await dynamoDb.scan(params).promise();

  return result;
};

const getArgumentById = async (argumentId: string) => {
  const params = {
    TableName: ARGUMENT_TABLE_NAME,
    KeyConditionExpression: 'argumentId=:' + argumentId,
    ExpressionAttributeValues: {
      ':argumentId': argumentId,
    },
  };

  const result = await dynamoDb.query(params).promise();

  return result;
};
```

```

return result.Items[0] as ArgumentTableEntry;
};

const deleteArgumentById = async (argumentId: string) => {
  const params = {
    TableName: ARGUMENT_TABLE_NAME,
    Key: {
      argumentId,
    },
  };
  const result = await dynamoDb.delete(params).promise();

  return result;
};

const addUserIdToArgumentUpvotes = async (
  argumentId: string,
  userId: string
) => {
  const currentArgument = await getArgumentById(argumentId);

  if (currentArgument.upvotedByUserIds.includes(userId)) {
    return {};
  }

  const params = {
    TableName: ARGUMENT_TABLE_NAME,
    Key: {
      argumentId,
    },
    UpdateExpression:
      'set #upvotedByUserIds = list_append(upvotedByUserIds, :userId)',
    ExpressionAttributeValues: {
      ':userId': [userId],
    },
  };

  const result = await dynamoDb.update(params).promise();

  return result;
};

```

```

};

const removeUserIdFromArgumentUpvotes = async (
  argumentId: string,
  userId: string
) => {
  const getParams: DynamoDB.DocumentClient.GetItemInput = {
    TableName: ARGUMENT_TABLE_NAME,
    Key: {
      argumentId,
    },
    ProjectionExpression: 'upvotedByUserIds',
  };

  const { Item } = await dynamoDb.get(getParams).promise();
  const upvotedByUserIds = Item.upvotedByUserIds as Array<string>;

  const indexToRemove = upvotedByUserIds.indexOf(userId);

  if (indexToRemove === -1) {
    return await getArgumentById(argumentId);
  }

  const updateParams: DynamoDB.DocumentClient.UpdateItemInput = {
    TableName: ARGUMENT_TABLE_NAME,
    Key: {
      argumentId,
    },
    UpdateExpression: 'REMOVE upvotedByUserIds[$indexToRemove]',
    ConditionExpression: 'upvotedByUserIds[$indexToRemove] = :userId',
    ExpressionAttributeValues: {
      ':userId': userId,
    },
  };

  const result = await dynamoDb.update(updateParams).promise();

  return result;
};

const addUserIdToArgumentDownvotes = async (
  argumentId: string,

```

```

userId: string
) => {
  const currentArgument = await getArgumentById(argumentId);

  if (currentArgument.downvotedByUserIds.includes(userId)) {
    return {};
  }

  const params = {
    TableName: ARGUMENT_TABLE_NAME,
    Key: {
      argumentId,
    },
    UpdateExpression:
      'set downvotedByUserIds = list_append(downvotedByUserIds, :userId)',
    ExpressionAttributeValues: {
      ':userId': [userId],
    },
  };
}

const result = await dynamoDb.update(params).promise();

return result;
};

const removeUserIdFromArgumentDownvotes = async (
  argumentId: string,
  userId: string
) => {
  const getParams: DynamoDB.DocumentClient.GetItemInput = {
    TableName: ARGUMENT_TABLE_NAME,
    Key: {
      argumentId,
    },
    ProjectionExpression: 'downvotedByUserIds',
  };

  const { Item } = await dynamoDb.get(getParams).promise();
  const downvotedByUserIds = Item.downvotedByUserIds as Array<string>;

  const indexToRemove = downvotedByUserIds.indexOf(userId);

```

```

if (indexToRemove === -1) {
  return await getArgumentById(argumentId);
}

const updateParams: DynamoDB.DocumentClient.UpdateItemInput = {
  TableName: ARGUMENT_TABLE_NAME,
  Key: {
    argumentId,
  },
  UpdateExpression: 'REMOVE downvotedByUserIds[${indexToRemove}]',
  ConditionExpression: 'downvotedByUserIds[${indexToRemove}] = :userId',
  ExpressionAttributeValues: {
    ':userId': userId,
  },
};

const result = await dynamoDb.update(updateParams).promise();

return result;
};

const addCommentIdToArgumentCommentIds = async (
  argumentId: string,
  commentId: string
) => {
  const currentArgument = await getArgumentById(argumentId);

  if (currentArgument.commentIds.includes(commentId)) {
    return {};
  }

  const params = {
    TableName: ARGUMENT_TABLE_NAME,
    Key: {
      argumentId,
    },
    UpdateExpression: 'set commentIds = list_append(commentIds, :commentId)',
    ExpressionAttributeValues: {
      ':commentId': [commentId],
    },
  };
}

```

```

const result = await dynamoDb.update(params).promise();

return result;
};

const removeCommentIdFromArgumentCommentIds = async (
  argumentId: string,
  commentId: string
) => {
  const getParams: DynamoDB.DocumentClient.GetItemInput = {
    TableName: ARGUMENT_TABLE_NAME,
    Key: {
      argumentId,
    },
    ProjectionExpression: 'commentIds',
  };

  const { Item } = await dynamoDb.get(getParams).promise();
  const commentIds = Item.commentIds as Array<string>;

  const indexToRemove = commentIds.indexOf(commentId);

  if (indexToRemove === -1) {
    return await getArgumentById(argumentId);
  }

  const updateParams: DynamoDB.DocumentClient.UpdateItemInput = {
    TableName: ARGUMENT_TABLE_NAME,
    Key: {
      argumentId,
    },
    UpdateExpression: 'REMOVE commentIds[$indexToRemove]',
    ConditionExpression: 'commentIds[$indexToRemove] = :commentId',
    ExpressionAttributeValues: {
      ':commentId': commentId,
    },
  };

  const result = await dynamoDb.update(updateParams).promise();

  return result;
};

```

```
export {  
  addCommentIdToArgumentCommentIds,  
  addUserIdToArgumentDownvotes,  
  addUserIdToArgumentUpvotes,  
  deleteArgumentById,  
  getArgumentById,  
  getArguments,  
  putArgument,  
  removeCommentIdFromArgumentCommentIds,  
  removeUserIdFromArgumentDownvotes,  
  removeUserIdFromArgumentUpvotes,  
};
```

## E.1.8 argupedia-api-master/src/adapters/database/commentTable.test.ts

```
import { DynamoDB } from 'aws-sdk';

import { config } from '../../helpers/DynamoDB/config';
import faker from '../../helpers/faker';
import { deleteCommentById, getCommentById, putComment } from './commentTable';

const { COMMENT_TABLE_NAME } = process.env;

const dynamoDb = new DynamoDB.DocumentClient(config);

it('shouldPutCommentIntoTable', async () => {
  const comment = faker.commentTableEntry();

  await putComment(comment);

  const { Item } = await dynamoDb
    .get({
      TableName: COMMENT_TABLE_NAME,
      Key: { commentId: comment.commentId },
    })
    .promise();

  expect(Item).toEqual(comment);
});

it('shouldGetCommentFromTableById', async () => {
  const comment = faker.commentTableEntry();

  await dynamoDb
    .put({ TableName: COMMENT_TABLE_NAME, Item: comment })
    .promise();

  const retrievedComment = await getCommentById(comment.commentId);

  expect(retrievedComment).toEqual(comment);
});

it('shouldDeleteCommentFromTable', async () => {
  const comment = faker.commentTableEntry();
```

```

await dynamoDb
  .put({ TableName: COMMENT_TABLE_NAME, Item: comment })
  .promise();

await deleteCommentById(comment.commentId);

const response = await dynamoDb
  .get({
    TableName: COMMENT_TABLE_NAME,
    Key: { commentId: comment.commentId },
  })
  .promise();

expect(response).toEqual({});

it('should_update_comment_in_table', async () => {
  const comment = faker.commentTableEntry();

  await dynamoDb
    .put({ TableName: COMMENT_TABLE_NAME, Item: comment })
    .promise();

  const newComment = faker.commentTableEntry({
    commentId: comment.commentId,
  });

  await putComment(newComment);

  const { Item } = await dynamoDb
    .get({
      TableName: COMMENT_TABLE_NAME,
      Key: { commentId: comment.commentId },
    })
    .promise();

  expect(Item).toEqual(newComment);
});

```

### E.1.9 argupedia-api-master/src/adapters/database/commentTable.ts

```
import { DynamoDB } from 'aws-sdk';

import { CommentTableEntry } from 'src/types/comments';

import { config } from '../../helpers/DynamoDB/config';

const dynamoDb = new DynamoDB.DocumentClient(config);

const { COMMENT_TABLE_NAME } = process.env;

const putComment = async (comment: CommentTableEntry) => {
  const params = {
    TableName: COMMENT_TABLE_NAME,
    Item: comment,
  };

  const result = await dynamoDb.put(params).promise();

  return result;
};

const getCommentById = async (commentId: string) => {
  const params = {
    TableName: COMMENT_TABLE_NAME,
    KeyConditionExpression: 'commentId=:commentId',
    ExpressionAttributeValues: {
      ':commentId': commentId,
    },
  };

  const { Items } = await dynamoDb.query(params).promise();

  return Items[0] as CommentTableEntry;
};

const deleteCommentById = async (commentId: string) => {
  const params = {
    TableName: COMMENT_TABLE_NAME,
    Key: {
      commentId,
    },
  };
}
```

```
};

const result = await dynamoDb.delete(params).promise();

return result;
};

export { deleteCommentById, getCommentById, putComment };
```

### E.1.10 argupedia-api-master/src/adapters/database/index.ts

```
import {
    addUserIdToArgumentDownvotes,
    addUserIdToArgumentUpvotes,
    deleteArgumentById,
    getArgumentById,
    removeUserIdFromArgumentDownvotes,
    removeUserIdFromArgumentUpvotes,
} from './argumentTable';

import { deleteCommentById, getCommentById } from './commentTable';

import {
    addArgumentIdForUserAuthoredArgumentIds,
    addArgumentIdForUserVotedArgumentIds,
    addCommentIdForUserAuthoredCommentIds,
    deleteUserById,
    getUserId,
    removeArgumentIdForUserAuthoredArgumentIds,
    removeArgumentIdForUserVotedArgumentIds,
    removeCommentIdForUserAuthoredCommentIds,
} from './userTable';

export default {
    addUserIdToArgumentDownvotes,
    addUserIdToArgumentUpvotes,
    deleteArgumentById,
    getArgumentById,
    removeUserIdFromArgumentDownvotes,
    removeUserIdFromArgumentUpvotes,
    addArgumentIdForUserAuthoredArgumentIds,
    addArgumentIdForUserVotedArgumentIds,
    addCommentIdForUserAuthoredCommentIds,
    deleteUserById,
    getUserId,
    removeArgumentIdForUserAuthoredArgumentIds,
    removeArgumentIdForUserVotedArgumentIds,
    removeCommentIdForUserAuthoredCommentIds,
    deleteCommentById,
    getCommentById,
};

};
```

### E.1.11 argupedia-api-master/src/adapters/database/userTable.test.ts

```
import { DynamoDB } from 'aws-sdk';

import { config } from '../../helpers/DynamoDB/config';
import faker from '../../helpers/faker';
import {

  addArgumentIdForUserAuthoredArgumentIds,
  addArgumentIdForUserVotedArgumentIds,
  addCommentIdForUserAuthoredCommentIds,
  deleteUserById,
  getUserById,
  putUser,
  removeArgumentIdForUserAuthoredArgumentIds,
  removeArgumentIdForUserVotedArgumentIds,
  removeCommentIdForUserAuthoredCommentIds,
} from './userTable';

const { USER_TABLE_NAME } = process.env;

const dynamoDb = new DynamoDB.DocumentClient(config);

it('shouldPutUserIntoTable', async () => {
  const user = faker.userTableEntry();

  await putUser(user);

  const { Item } = await dynamoDb
    .get({
      TableName: USER_TABLE_NAME,
      Key: { userId: user.userId },
    })
    .promise();

  expect(Item).toEqual(user);
});

it('shouldGetUserById', async () => {
  const user = faker.userTableEntry();

  await dynamoDb.put({ TableName: USER_TABLE_NAME, Item: user }).promise();
```

```

const retrievedUser = await getUserById(user.userId);

expect(retrievedUser).toEqual(user);
});

it('shouldDeleteUserById', async () => {
  const user = faker.userTableEntry();

  await dynamoDb.put({ TableName: USER_TABLE_NAME, Item: user }).promise();

  await deleteUserById(user.userId);

  const response = await dynamoDb
    .get({
      TableName: USER_TABLE_NAME,
      Key: { userId: user.userId },
    })
    .promise();

  expect(response).toEqual({});
});

it('shouldUpdateUserInTable', async () => {
  const user = faker.userTableEntry();

  await dynamoDb.put({ TableName: USER_TABLE_NAME, Item: user }).promise();

  const newUser = faker.userTableEntry({
    userId: user.userId,
  });

  await putUser(newUser);

  const { Item } = await dynamoDb
    .get({
      TableName: USER_TABLE_NAME,
      Key: { userId: newUser.userId },
    })
    .promise();

  expect(Item).toEqual(newUser);
});

```

```

it('should add argument id to user authored argument ids', async () => {
  const user = faker.userTableEntry();

  await dynamoDb.put({ TableName: USER_TABLE_NAME, Item: user }).promise();

  const argumentIdToAdd = faker.datatype.uuid();

  await addArgumentIdForUserAuthoredArgumentIds(user.userId, argumentIdToAdd);

  const { Item } = await dynamoDb
    .get({
      TableName: USER_TABLE_NAME,
      Key: { userId: user.userId },
    })
    .promise();

  expect(Item.authoredArgumentIds).toContain(argumentIdToAdd);
});

it('should remove argument id from user authored argument ids', async () => {
  const argumentIdToBeRemoved = faker.datatype.uuid();

  const user = faker.userTableEntry({
    authoredArgumentIds: [
      ...Array.from({ length: faker.datatype.number(20) }, () =>
        faker.datatype.uuid()
      ),
      argumentIdToBeRemoved,
    ],
  });

  await dynamoDb.put({ TableName: USER_TABLE_NAME, Item: user }).promise();

  await removeArgumentIdForUserAuthoredArgumentIds(
    user.userId,
    argumentIdToBeRemoved
  );

  const { Item } = await dynamoDb
    .get({
      TableName: USER_TABLE_NAME,
    })
    .promise();
});

```

```

    Key: { userId: user.userId },
  })
  .promise();

expect(Item.authoredArgumentIds).not.toContain(argumentIdToBeRemoved);
});

it('should_add_argument_id_to_user_authored_comment_ids', async () => {
  const user = faker.userTableEntry();

  await dynamoDb.put({ TableName: USER_TABLE_NAME, Item: user }).promise();

  const commentIdToAdd = faker.datatype.uuid();

  await addCommentIdForUserAuthoredCommentIds(user.userId, commentIdToAdd);

  const { Item } = await dynamoDb
    .get({
      TableName: USER_TABLE_NAME,
      Key: { userId: user.userId },
    })
    .promise();

  expect(Item.authoredCommentIds).toContain(commentIdToAdd);
});

it('should_remove_argument_id_from_user_authored_comment_ids', async () => {
  const commentIdToBeRemoved = faker.datatype.uuid();

  const user = faker.userTableEntry({
    authoredCommentIds: [
      ...Array.from({ length: faker.datatype.number(20) }, () =>
        faker.datatype.uuid()
      ),
      commentIdToBeRemoved,
    ],
  });
  await dynamoDb.put({ TableName: USER_TABLE_NAME, Item: user }).promise();

  await removeCommentIdForUserAuthoredCommentIds(
    user.userId,
  );
});

```

```

commentIdToBeRemoved
);

const { Item } = await dynamoDb
.get({
  TableName: USER_TABLE_NAME,
  Key: { userId: user.userId },
})
.promise();

expect(Item.authoredCommentIds).not.toContain(commentIdToBeRemoved);
});

it('should_add_argument_id_to_user_voted_argument_ids', async () => {
  const user = faker.userTableEntry();

  await dynamoDb.put({ TableName: USER_TABLE_NAME, Item: user }).promise();

  const argumentIdToAdd = faker.datatype.uuid();

  await addArgumentIdForUserVotedArgumentIds(user.userId, argumentIdToAdd);

  const { Item } = await dynamoDb
.get({
  TableName: USER_TABLE_NAME,
  Key: { userId: user.userId },
})
.promise();

expect(Item.votedArgumentIds).toContain(argumentIdToAdd);
});

it('should_remove_argument_id_from_user_voted_argument_ids', async () => {
  const argumentIdToBeRemoved = faker.datatype.uuid();

  const user = faker.userTableEntry({
    votedArgumentIds: [
      ...Array.from({ length: faker.datatype.number(20) }, () =>
        faker.datatype.uuid()
      ),
      argumentIdToBeRemoved,
    ],
  });

```

```
});  
  
await dynamoDb.put({ TableName: USER_TABLE_NAME, Item: user }).promise();  
  
await removeArgumentIdForUserVotedArgumentIds(  
  user.userId,  
  argumentIdToBeRemoved  
);  
  
const { Item } = await dynamoDb  
  .get({  
    TableName: USER_TABLE_NAME,  
    Key: { userId: user.userId },  
  })  
  .promise();  
  
expect(Item.votedArgumentIds).not.toContain(argumentIdToBeRemoved);  
});
```

### E.1.12 argupedia-api-master/src/adapters/database/userTable.ts

```
import { DynamoDB } from 'aws-sdk';

import { UserTableEntry } from 'src/types/users';

import { config } from '../../helpers/DynamoDB/config';

const dynamoDb = new DynamoDB.DocumentClient(config);

const { USER_TABLE_NAME } = process.env;

const putUser = async (user: UserTableEntry) => {
  const params = {
    TableName: USER_TABLE_NAME,
    Item: user,
  };

  const result = await dynamoDb.put(params).promise();

  return result;
};

const getUserById = async (userId: string) => {
  const params = {
    TableName: USER_TABLE_NAME,
    KeyConditionExpression: 'userId=:_userId',
    ExpressionAttributeValues: {
      ':userId': userId,
    },
  };

  const result = await dynamoDb.query(params).promise();

  return result.Items[0] as UserTableEntry;
};

const deleteUserById = async (userId: string) => {
  const params = {
    TableName: USER_TABLE_NAME,
    Key: {
      userId,
    },
  };
}
```

```

};

const result = await dynamoDb.delete(params).promise();

return result;
};

const addArgumentIdForUserAuthoredArgumentIds = async (
  userId: string,
  argumentId: string
) => {
  const currentUser = await getUserById(userId);

  if (currentUser === undefined) {
    throw new Error('User does not exist');
  }

  if (currentUser.authoredArgumentIds.includes(argumentId)) {
    return {};
  }

  const params = {
    TableName: USER_TABLE_NAME,
    Key: {
      userId,
    },
    UpdateExpression:
      'set authoredArgumentIds=list_append(authoredArgumentIds,:argumentId)',
    ExpressionAttributeValues: {
      ':argumentId': [argumentId],
    },
  };

  const result = await dynamoDb.update(params).promise();

  return result;
};

const removeArgumentIdForUserAuthoredArgumentIds = async (
  userId: string,
  argumentId: string
) => {

```

```

const getParams: DynamoDB.DocumentClient.GetItemInput = {
  TableName: USER_TABLE_NAME,
  Key: {
    userId,
  },
  ProjectionExpression: 'authoredArgumentIds',
};

const { Item } = await dynamoDb.get(getParams).promise();

if (Item === undefined) {
  throw new Error('User does not exist');
}

const authoredArgumentIds = Item.authoredArgumentIds as Array<string>;
const indexToRemove = authoredArgumentIds.indexOf(argumentId);

if (indexToRemove === -1) {
  return await getUserById(userId);
}

const updateParams: DynamoDB.DocumentClient.UpdateItemInput = {
  TableName: USER_TABLE_NAME,
  Key: {
    userId,
  },
  UpdateExpression: 'REMOVE authoredArgumentIds[${indexToRemove}]',
  ConditionExpression: 'authoredArgumentIds[${indexToRemove}] = :argumentId',
  ExpressionAttributeValues: {
    ':argumentId': argumentId,
  },
};

const result = await dynamoDb.update(updateParams).promise();

return result;
};

const addCommentIdForUserAuthoredCommentIds = async (
  userId: string,
  commentId: string
)

```

```

) => {

  const currentUser = await getUserById(userId);

  if (currentUser === undefined) {
    throw new Error('User does not exist');
  }

  if (currentUser.authoredCommentIds.includes(commentId)) {
    return {};
  }

  const params = {
    TableName: USER_TABLE_NAME,
    Key: {
      userId,
    },
    UpdateExpression:
      'set authoredCommentIds = list_append(authoredCommentIds, :commentId)',
    ExpressionAttributeValues: {
      ':commentId': [commentId],
    },
  };

  const result = await dynamoDb.update(params).promise();

  return result;
};

const removeCommentIdForUserAuthoredCommentIds = async (
  userId: string,
  commentId: string
) => {
  const getParams: DynamoDB.DocumentClient.GetItemInput = {
    TableName: USER_TABLE_NAME,
    Key: {
      userId,
    },
    ProjectionExpression: 'authoredCommentIds',
  };

  const { Item } = await dynamoDb.get(getParams).promise();

```

```

if (Item === undefined) {
  throw new Error('User does not exist');
}

const authoredCommentIds = Item.authoredCommentIds as Array<string>;

const indexToRemove = authoredCommentIds.indexOf(commentId);

if (indexToRemove === -1) {
  return await getUserById(userId);
}

const updateParams: DynamoDB.DocumentClient.UpdateItemInput = {
  TableName: USER_TABLE_NAME,
  Key: {
    userId,
  },
  UpdateExpression: 'REMOVE authoredCommentIds[${{indexToRemove}}]',
  ConditionExpression: 'authoredCommentIds[${{indexToRemove}}] = :commentId',
  ExpressionAttributeValues: {
    ':commentId': commentId,
  },
};

const result = await dynamoDb.update(updateParams).promise();

return result;
};

const addArgumentIdForUserVotedArgumentIds = async (
  userId: string,
  argumentId: string
) => {
  const currentUser = await getUserById(userId);

  if (currentUser === undefined) {
    throw new Error('User does not exist');
  }

  if (currentUser.votedArgumentIds.includes(argumentId)) {
    return {};
  }
}

```

```

const params = {
  TableName: USER_TABLE_NAME,
  Key: {
    userId,
  },
  UpdateExpression:
    'set_votedArgumentIds=list_append(votedArgumentIds,:argumentId)',
  ExpressionAttributeValues: {
    ':argumentId': [argumentId],
  },
};

const result = await dynamoDb.update(params).promise();

return result;
};

const removeArgumentIdForUserVotedArgumentIds = async (
  userId: string,
  argumentId: string
) => {
  const getParams: DynamoDB.DocumentClient.GetItemInput = {
    TableName: USER_TABLE_NAME,
    Key: {
      userId,
    },
    ProjectionExpression: 'votedArgumentIds',
  };

  const { Item } = await dynamoDb.get(getParams).promise();

  if (Item === undefined) {
    throw new Error('User does not exist');
  }

  const votedArgumentIds = Item.votedArgumentIds as Array<string>;
  const indexToRemove = votedArgumentIds.indexOf(argumentId);

  if (indexToRemove === -1) {
    return await getUserById(userId);
  }
};

```

```

    }

const updateParams: DynamoDB.DocumentClient.UpdateItemInput = {
  TableName: USER_TABLE_NAME,
  Key: {
    userId,
  },
  UpdateExpression: 'REMOVE votedArgumentIds[${indexToRemove}]',
  ConditionExpression: 'votedArgumentIds[${indexToRemove}] = :argumentId',
  ExpressionAttributeValues: {
    ':argumentId': argumentId,
  },
};

const result = await dynamoDb.update(updateParams).promise();

return result;
};

export {
  addArgumentIdForUserAuthoredArgumentIds,
  addArgumentIdForUserVotedArgumentIds,
  addCommentIdForUserAuthoredCommentIds,
  deleteUserById,
  getUserById,
  putUser,
  removeArgumentIdForUserAuthoredArgumentIds,
  removeArgumentIdForUserVotedArgumentIds,
  removeCommentIdForUserAuthoredCommentIds,
};

```

### E.1.13 argupedia-api-master/src/functions/clearvoteArgumentWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import {
    addUserIdToArgumentDownvotes,
    addUserIdToArgumentUpvotes,
    getArgumentById,
    putArgument,
} from '@adapters/database/argumentTable';

import { getUserById, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../helpers/faker';

jest.mock('@libs/lambda');

describe('clearvoteArgumentWithId', () => {
    let main;
    let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

    beforeAll(async () => {
        mockedMiddyfy = mocked(middyfy);
        mockedMiddyfy.mockImplementation((handler: Handler) => {
            return handler as never;
        });

        main = (await import('../handler')).main;
    });

    afterAll(() => {
        jest.resetModules();
    });

    it('should_remove_user_id_from_argument_downvotes', async () => {
        const userTableEntry = faker.userTableEntry();

        await putUser(userTableEntry);
```

```

const argumentTableEntry = faker.argumentTableEntry({
  downvotedByUserIds: [userTableEntry.userId],
});

await putArgument(argumentTableEntry);
await addUserIdToArgumentDownvotes(
  argumentTableEntry.argumentId,
  userTableEntry.userId
);

expect(argumentTableEntry.downvotedByUserIds).toContain(
  userTableEntry.userId
);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await main(event);

const updatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(updatedArgumentTableEntry.downvotedByUserIds).not.toContain(
  userTableEntry.userId
);

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.votedArgumentIds).not.toContain(
  argumentTableEntry.argumentId
);
};

it('should_remove_user_id_from_argument_upvotes', async () => {

```

```

const userTableEntry = faker.userTableEntry();

await putUser(userTableEntry);

const argumentTableEntry = faker.argumentTableEntry({
  upvotedByUserIds: [userTableEntry.userId],
});

await putArgument(argumentTableEntry);
await addUserIdToArgumentUpvotes(
  argumentTableEntry.argumentId,
  userTableEntry.userId
);

expect(argumentTableEntry.upvotedByUserIds).toContain(
  userTableEntry.userId
);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await main(event);

const updatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(updatedArgumentTableEntry.upvotedByUserIds).not.toContain(
  userTableEntry.userId
);

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.votedArgumentIds).not.toContain(
  argumentTableEntry.argumentId
);

```

```
});

it('should_reject_any_operation_where_argumentId_is_not_valid', async () => {
  const event = {
    pathParameters: {
      argumentId: faker.datatype.uuid(),
    },
    headers: {
      'mocked-user-id': faker.datatype.uuid(),
    },
  };

  await expect(main(event)).rejects.toThrow(
    'No argument with specified argumentId'
  );
});

});
```

## E.1.14 argupedia-api-master/src/functions/clearvoteArgumentWithId/handler.ts

```
import { removeArgumentIdForUserVotedArgumentIds } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';

import {
  getArgumentById,
  removeUserIdFromArgumentDownvotes,
  removeUserIdFromArgumentUpvotes,
} from '../../../../../adapters/database/argumentTable';
import schema from './schema';

const clearvoteArgumentWithId: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  const argumentId = event.pathParameters.argumentId;

  const argumentTableEntry = await getArgumentById(argumentId);

  if (argumentTableEntry === undefined) {
    throw new Error('No argument with specified argumentId');
  }

  await removeUserIdFromArgumentUpvotes(argumentId, userId);
  await removeUserIdFromArgumentDownvotes(argumentId, userId);
  await removeArgumentIdForUserVotedArgumentIds(userId, argumentId);
```

```
const argumentToBeReturned = await getArgumentById(argumentId);

return formatJSONResponse({
  message: argumentToBeReturned,
  event,
});

export const main = middyfy(clearvoteArgumentWithId);
```

### E.1.15 argupedia-api-master/src/functions/clearvoteArgumentWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'post',
        path: 'rest/api/1/argument/{argumentId}/clearvote',
        cors: true,
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.16 argupedia-api-master/src/functions/clearvoteArgumentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.17 argupedia-api-master/src/functions/clearvoteArgumentWithId/schema.ts

```
export default {} as const;
```

## E.1.18 argupedia-api-master/src/functions/createCommentForArgumentWithId/handler.test.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { getArgumentById, putArgument } from '@adapters/database/argumentTable';
import { getUserId, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { CommentTableEntry } from 'src/types/comments';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('createCommentForArgumentWithArgumentId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeAll(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should_add_new_comment_for_argument', async () => {
    const userTableEntry = faker.userTableEntry();

    await putUser(userTableEntry);

    const argumentTableEntry = faker.argumentTableEntry();

    await putArgument(argumentTableEntry);
  });
});
```

```

const event = {
  body: {
    commentBody: faker.random.words(30),
  },
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as CommentTableEntry;

expect(entry.commentBody).toEqual(event.body.commentBody);

const updatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(updatedArgumentTableEntry.commentIds).toContain(entry.commentId);

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.authoredCommentIds).toContain(entry.commentId);
});

it('should reject any operation where argumentId is not valid', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const event = {
    body: {
      commentBody: faker.random.words(30),
    },
    pathParameters: {
      argumentId: faker.datatype.uuid(),
    },
  };
}

```

```
headers: {  
  'mocked-user-id': userTableEntry.userId,  
},  
};  
  
await expect(main(event)).rejects.toThrow(  
  'No argument with specified argumentId'  
);  
});  
});
```

### E.1.19 argupedia-api-master/src/functions/createCommentForArgumentWithId/handler

```
import { putComment } from '@adapters/database/commentTable';
import { addCommentIdForUserAuthoredCommentIds } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';
import { CommentTableEntry } from 'src/types/comments';
import { v4 as uuidv4 } from 'uuid';

import {
  addCommentIdToArgumentCommentIds,
  getArgumentById,
} from '../../../../../adapters/database/argumentTable';
import schema from './schema';

const createCommentForArgumentWithId: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  const originalArgument = await getArgumentById(
    event.pathParameters.argumentId
  );

  if (originalArgument === undefined) {
    throw new Error('No argument with specified argumentId');
  }

  const createdDate = new Date().toISOString();
```

```

const commentTableEntry: CommentTableEntry = {
  commentId: uuidv4(),
  argumentId: event.pathParameters.argumentId,
  authorId: userId,
  commentBody: event.body.commentBody,
  createdDate: createdDate,
  modifiedDate: createdDate,
};

await putComment(commentTableEntry);

await addCommentIdToArgumentCommentIds(
  event.pathParameters.argumentId,
  commentTableEntry.commentId
);

await addCommentIdForUserAuthoredCommentIds(
  userId,
  commentTableEntry.commentId
);

return formatJSONResponse({
  message: commentTableEntry,
  event,
});
};

export const main = middyfy(createCommentForArgumentWithId);

```

## E.1.20 argupedia-api-master/src/functions/createCommentForArgumentWithId/index.js

```
import { handlerPath } from '@libs/handlerResolver';

import schema from './schema';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'post',
        path: 'rest/api/1/argument/{argumentId}/comments',
        cors: true,
        request: {
          schemas: {
            'application/json': schema,
          },
        },
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    ],
  },
};
```

### E.1.21 argupedia-api-master/src/functions/createCommentForArgumentWithId/moc

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

## E.1.22 argupedia-api-master/src/functions/createCommentForArgumentWithId/sche

```
export default {
  type: 'object',
  properties: {
    commentBody: { type: 'string' },
  },
  required: ['commentBody'],
} as const;
```

### E.1.23 argupedia-api-master/src/functions/createNewArgument/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { getUserById, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

jest.mock('@libs/lambda');

describe('createNewArgument', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
  });

  main = (await import('./handler')).main;
});

afterAll(() => {
  jest.resetModules();
});

it('should_add_new_action_argument', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const event = {
    body: {
      argumentType: 'ActionArgument',
      argument: faker.actionArgument(),
    }
  }
```

```

    },
    headers: {
      'mocked-user-id': userTableEntry.userId,
    },
  };

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as ArgumentTableEntry;

expect(entry.argument).toEqual(event.body.argument);
expect(entry.argumentType).toEqual(event.body.argumentType);

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.authoredArgumentIds).toContain(
  entry.argumentId
);
});

it('should_reject_any_operation_where_argumentType_is_not_valid', async () => {
  const event = {
    body: {
      argumentType: 'InvalidArgumentType',
      argument: faker.argument(),
    },
    headers: {
      'mocked-user-id': faker.datatype.uuid(),
    },
  };

  await expect(main(event)).rejects.toThrow('Invalid_argument_type');
});

it('should_reject_any_operation_where_argumentType_does_not_match_provided_argument', async () => {
  const event = {
    body: {
      argumentType: 'ActionArgument',
      argument: faker.argumentForArgumentType(faker.attackingArgumentType()),
    },
    headers: {
      'mocked-user-id': faker.datatype.uuid(),
    },
  };
});

```

```
    },  
    );  
  
    await expect(main(event)).rejects.toThrow(  
      'argument does not match argumentType'  
    );  
  );  
});
```

## E.1.24 argupedia-api-master/src/functions/createNewArgument/handler.ts

```
import { addArgumentIdForUserAuthoredArgumentIds } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';
import { v4 as uuidv4 } from 'uuid';

import { putArgument } from '../../adapters/database/argumentTable';
import { labelArguments, validateArgument } from '../../../../../helpers/argupedia';
import {
    Argument,
    ArgumentTableEntry,
    CreateNewArgumentHandlerBody,
} from '../../types/arguments';
import schema from './schema';

const createNewArgument: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    let payload: CognitoIdTokenPayload = undefined;

    if (
        event.headers !== undefined &&
        event.headers.Authorization !== undefined
    ) {
        payload = await verifyJwtToken(event.headers.Authorization.substring(7));
    }

    const isTest = process.env.JEST_WORKER_ID;
    const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

    validateArgument(event.body as CreateNewArgumentHandlerBody);

    const createdDate = new Date().toISOString();

    const argument = event.body.argument as Argument;

    const argumentTableEntry: ArgumentTableEntry = {
        argumentId: uuidv4(),
        authorId: userId,
        content: argument.content,
        dateCreated: createdDate,
        dateModified: createdDate,
        title: argument.title,
        type: argument.type,
    };

    const result = await putArgument(argumentTableEntry);
    return formatJSONResponse(result);
}
```

```

argumentType: event.body.argumentType,
argument: argument,
argumentText: event.body.argumentText,
argumentDepth: 0,
authorId: userId,
createdDate: createdDate,
modifiedDate: createdDate,
attacksArgumentId: undefined,
attackedByArgumentIds: new Array<string>(),
supportsArgumentId: undefined,
supportedByArgumentIds: new Array<string>(),
upvotedByUserIds: new Array<string>(),
downvotedByUserIds: new Array<string>(),
label: 'IN',
commentIds: new Array<string>(),
};

await putArgument(argumentTableEntry);

await labelArguments(argumentTableEntry.argumentId);

await addArgumentIdForUserAuthoredArgumentIds(
  userId,
  argumentTableEntry.argumentId
);

return formatJSONResponse({
  message: argumentTableEntry,
  event,
});
};

export const main = middyfy(createNewArgument);

```

## E.1.25 argupedia-api-master/src/functions/createNewArgument/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

import schema from './schema';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'post',
        path: 'rest/api/1/argument/',
        cors: true,
        request: {
          schemas: {
            'application/json': schema,
          },
        },
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    ],
  };
};
```

### E.1.26 argupedia-api-master/src/functions/createNewArgument/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.27 argupedia-api-master/src/functions/createNewArgument/schema.ts

```
export default {  
  type: 'object',  
  properties: {  
    argumentType: { type: 'string' },  
    argument: { type: 'object' },  
    argumentText: { type: 'string' },  
  },  
  required: ['argumentType', 'argument'],  
} as const;
```

## E.1.28 argupedia-api-master/src/functions/createNewAttackingArgument/handler.sp

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { getArgumentById, putArgument } from '@adapters/database/argumentTable';
import { getUserId, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { DynamoDB } from 'aws-sdk';
import { mocked } from 'ts-jest/utils';

import { config } from '../../helpers/DynamoDB/config';
import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

const dynamoDb = new DynamoDB.DocumentClient(config);
const { ARGUMENT_TABLE_NAME } = process.env;

jest.mock('@libs/lambda');

describe('createNewAttackingArgument', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
  });

  main = (await import('./handler')).main;
});

afterAll(() => {
  jest.resetModules();
});

it('should_add_new_attacking_argument', async () => {
  const userTableEntry = faker.userTableEntry();
```

```

await putUser(userTableEntry);

const attackedArgument = faker.argumentTableEntry({
  attackedByArgumentIds: [],
  supportedByArgumentIds: [],
});

await putArgument(attackedArgument);

const argumentType = faker.attackingArgumentType();

const event = {
  body: {
    argumentType: argumentType,
    argument: faker.argumentForArgumentType(argumentType),
  },
  pathParameters: {
    argumentId: attackedArgument.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as ArgumentTableEntry;

expect(entry.argument).toEqual(event.body.argument);
expect(entry.argumentType).toEqual(event.body.argumentType);
expect(entry.attacksArgumentId).toEqual(event.pathParameters.argumentId);

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.authoredArgumentIds).toContain(
  entry.argumentId
);

const updatedAttackedArgument = await getArgumentById(
  attackedArgument.argumentId
);

```

```

expect(updatedAttackedArgument.attackedByArgumentIds).toContain(
  entry.argumentId
);
});

it('should_reject_any_operation_where_argumentType_is_not_valid', async () => {
  const event = {
    body: {
      argumentType: 'InvalidArgumentType',
      argument: faker.argument(),
    },
    headers: {
      'mocked-user-id': faker.datatype.uuid(),
    },
  };
}

await expect(main(event)).rejects.toThrow('InvalidArgumentType');
});

it('should_reject_any_operation_where_argumentType_does_not_match_provided_argument', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const attackedArgument = faker.argumentTableEntry({
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
  });

  await putArgument(attackedArgument);

  const event = {
    body: {
      argumentType: 'ActionArgument',
      argument: faker.argumentForArgumentType(faker.attackingArgumentType()),
    },
    pathParameters: {
      argumentId: attackedArgument.argumentId,
    },
    headers: {
      'mocked-user-id': userTableEntry.userId,
    },
  };
}

```

```

};

await expect(main(event)).rejects.toThrow(
  'argument does not match argumentType'
);
});

it('should label parent arguments correctly', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const attackedArgument = faker.argumentTableEntry({
    attackedByArgumentIds: [],
    supportedByArgumentIds: []
  });

  await putArgument(attackedArgument);

  const parentEvent = {
    body: {
      argumentType: 'ActionArgument',
      argument: faker.actionArgument(),
    },
    pathParameters: {
      argumentId: attackedArgument.argumentId,
    },
    headers: {
      'mocked-user-id': userTableEntry.userId,
    },
  };
}

const parent = await main(parentEvent);
const parentResponse = JSON.parse(parent.body);
const parentResponseMessage = parentResponse.message as ArgumentTableEntry;

expect(parentResponseMessage.label).toBe('IN');

const argumentType = faker.attackingArgumentType();

const event = {
  body: {

```

```

argumentType: argumentType,
argument: faker.argumentForArgumentType(argumentType),
},
pathParameters: {
argumentId: parentResponseMessage.argumentId,
},
headers: {
'mocked-user-id': userTableEntry.userId,
},
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as ArgumentTableEntry;

expect(entry.label).toEqual('IN');

const { Item } = await dynamoDb
.get({
TableName: ARGUMENT_TABLE_NAME,
Key: { argumentId: parentResponseMessage.argumentId },
})
.promise();

expect(Item.label).toBe('OUT');
});

it('should add argument id to attackedByArgumentId attribute of parent argument', async () => {
const userTableEntry = faker.userTableEntry();

await putUser(userTableEntry);

const attackedArgument = faker.argumentTableEntry({
attackedByArgumentIds: [],
supportedByArgumentIds: [],
});

await putArgument(attackedArgument);

const parentEvent = {
body: {
argumentType: 'ActionArgument',

```

```

    argument: faker.actionArgument(),
},
pathParameters: {
  argumentId: attackedArgument.argumentId,
},
headers: {
  'mocked-user-id': userTableEntry.userId,
},
};

const parent = await main(parentEvent);
const parentResponse = JSON.parse(parent.body);
const parentResponseMessage = parentResponse.message as ArgumentTableEntry;

const child1ArgumentType = faker.attackingArgumentType();
const child1Event = {
  body: {
    argumentType: child1ArgumentType,
    argument: faker.argumentForArgumentType(child1ArgumentType),
  },
  pathParameters: {
    argumentId: parentResponseMessage.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};
};

const child1 = await main(child1Event);
const child1Response = JSON.parse(child1.body);
const child1ResponseMessage = child1Response.message as ArgumentTableEntry;

const child2ArgumentType = faker.attackingArgumentType();
const child2Event = {
  body: {
    argumentType: child2ArgumentType,
    argument: faker.argumentForArgumentType(child2ArgumentType),
  },
  pathParameters: {
    argumentId: parentResponseMessage.argumentId,
  },
  headers: {

```

```

'mocked-user-id': userTableEntry.userId,
},
};

const child2 = await main(child2Event);
const child2Response = JSON.parse(child2.body);
const child2ResponseMessage = child2Response.message as ArgumentTableEntry;

const { Item } = await dynamoDb
.get({
  TableName: ARGUMENT_TABLE_NAME,
  Key: { argumentId: parentResponseMessage.argumentId },
})
.promise();

expect(Item.attackedByArgumentIds).toContain(
  child1ResponseMessage.argumentId
);
expect(Item.attackedByArgumentIds).toContain(
  child2ResponseMessage.argumentId
);
});

it('should_only_attack_root_argument_of_supporting_argument_subtree', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const argumentA = faker.argumentTableEntry({
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
  });

  const argumentB = faker.argumentTableEntry({
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
    supportsArgumentId: argumentA.argumentId,
  });

  const argumentC = faker.argumentTableEntry({
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
    supportsArgumentId: argumentB.argumentId,
  });
});

```

```

argumentA.attackedByArgumentIds = [argumentB.argumentId];
argumentB.attackedByArgumentIds = [argumentC.argumentId];

await putArgument(argumentA);
await putArgument(argumentB);
await putArgument(argumentC);

const argumentType = faker.attackingArgumentType();
const event = {
  body: {
    argumentType: argumentType,
    argument: faker.argumentForArgumentType(argumentType),
  },
  pathParameters: {
    argumentId: argumentC.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as ArgumentTableEntry;

expect(entry.attacksArgumentId).toEqual(argumentA.argumentId);
});

});

```

## E.1.29 argupedia-api-master/src/functions/createNewAttackingArgument/handler.ts

```
import { addArgumentIdForUserAuthoredArgumentIds } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';
import { v4 as uuidv4 } from 'uuid';

import {
  getArgumentById,
  putArgument,
} from '../../adapters/database/argumentTable';
import { labelArguments, validateArgument } from '../../../../../helpers/argupedia';
import {
  Argument,
  ArgumentTableEntry,
  CreateNewArgumentHandlerBody,
} from '../../types/arguments';
import schema from './schema';

const getRootArgumentOfSupportingArgumentTree = async (
  argumentId: string
): Promise<ArgumentTableEntry> => {
  const argument = await getArgumentById(argumentId);

  if (argument.supportsArgumentId === undefined) {
    return argument;
  } else {
    return await getRootArgumentOfSupportingArgumentTree(
      argument.supportsArgumentId
    );
  }
};

const createNewAttackingArgument: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
```

```

event.headers !== undefined &&
event.headers.Authorization !== undefined
) {
  payload = await verifyJwtToken(event.headers.Authorization.substring(7));
}

const isTest = process.env.JEST_WORKER_ID;
const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

validateArgument(event.body as CreateNewArgumentHandlerBody);

if (
  event.pathParameters === undefined ||
  event.pathParameters.argumentId === undefined
) {
  throw new Error('No argumentId specified');
}

const attacksArgument = await getRootArgumentOfSupportingArgumentTree(
  event.pathParameters.argumentId
);

const createdDate = new Date().toISOString();

const argument = event.body.argument as Argument;

const argumentTableEntry: ArgumentTableEntry = {
  argumentId: uuidv4(),
  argumentType: event.body.argumentType,
  argument: argument,
  argumentText: event.body.argumentText,
  argumentDepth: attacksArgument.argumentDepth + 1,
  authorId: userId,
  createdDate: createdDate,
  modifiedDate: createdDate,
  attacksArgumentId: attacksArgument.argumentId,
  attackedByArgumentIds: new Array<string>(),
  supportedByArgumentIds: new Array<string>(),
  upvotedByUserIds: new Array<string>(),
  downvotedByUserIds: new Array<string>(),
  label: 'IN',
  commentIds: new Array<string>(),
}

```

```
};

await putArgument(argumentTableEntry);

await putArgument({
  ...attacksArgument,
  attackedByArgumentIds: attacksArgument.attackedByArgumentIds.concat(
    argumentTableEntry.argumentId
  ),
});

await labelArguments(argumentTableEntry.argumentId);

await addArgumentIdForUserAuthoredArgumentIds(
  userId,
  argumentTableEntry.argumentId
);

return formatJSONResponse({
  message: argumentTableEntry,
  event,
});
};

export const main = middyfy(createNewAttackingArgument);
```

### E.1.30 argupedia-api-master/src/functions/createNewAttackingArgument/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

import schema from './schema';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'post',
        path: 'rest/api/1/argument/{argumentId}/attacking',
        cors: true,
        request: {
          schemas: {
            'application/json': schema,
          },
        },
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.31 argupedia-api-master/src/functions/createNewAttackingArgument/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

### E.1.32 argupedia-api-master/src/functions/createNewAttackingArgument/schema.ts

```
export default {
  type: 'object',
  properties: {
    argumentType: { type: 'string' },
    argument: { type: 'object' },
    argumentText: { type: 'string' },
  },
  required: ['argumentType', 'argument'],
} as const;
```

### E.1.33 argupedia-api-master/src/functions/createNewSupportingArgument/handler.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { getArgumentById, putArgument } from '@adapters/database/argumentTable';
import { getUserId, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { DynamoDB } from 'aws-sdk';
import { mocked } from 'ts-jest/utils';

import { config } from '../../helpers/DynamoDB/config';
import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

const dynamoDb = new DynamoDB.DocumentClient(config);
const { ARGUMENT_TABLE_NAME } = process.env;

jest.mock('@libs/lambda');

describe('createNewAttackingArgument', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
  });

  main = (await import('./handler')).main;
});

afterAll(() => {
  jest.resetModules();
});

it('should_add_new_supporting_argument', async () => {
  const userTableEntry = faker.userTableEntry();
```

```

await putUser(userTableEntry);

const supportedArgument = faker.argumentTableEntry({
  attackedByArgumentIds: [],
  supportedByArgumentIds: [],
});

await putArgument(supportedArgument);

const argumentType = faker.attackingArgumentType();

const event = {
  body: {
    argumentType: argumentType,
    argument: faker.argumentForArgumentType(argumentType),
  },
  pathParameters: {
    argumentId: supportedArgument.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as ArgumentTableEntry;

expect(entry.argument).toEqual(event.body.argument);
expect(entry.argumentType).toEqual(event.body.argumentType);
expect(entry.supportsArgumentId).toEqual(event.pathParameters.argumentId);

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.authoredArgumentIds).toContain(
  entry.argumentId
);

const updatedSupportedArgument = await getArgumentById(
  supportedArgument.argumentId
);

```

```

expect(entry.label).toEqual(updatedSupportedArgument.label);

expect(updatedSupportedArgument.supportedByArgumentIds).toContain(
  entry.argumentId
);
});

it('should_reject_any_operation_where_argumentType_is_not_valid', async () => {
  const supportedArgument = faker.argumentTableEntry({
    attackedByArgumentIds: [],
    supportedByArgumentIds: []
  });

  await putArgument(supportedArgument);

  const event = {
    body: {
      argumentType: 'InvalidArgumentType',
      argument: faker.argument(),
    },
    pathParameters: {
      argumentId: supportedArgument.argumentId,
    },
    headers: {
      'mocked-user-id': faker.datatype.uuid(),
    },
  };
}

await expect(main(event)).rejects.toThrow('InvalidArgumentType');
});

it('should_reject_any_operation_where_argumentType_does_not_match_provided_argument', async () => {
  const supportedArgument = faker.argumentTableEntry({
    attackedByArgumentIds: [],
    supportedByArgumentIds: []
  });

  await putArgument(supportedArgument);

  const event = {
    body: {
      argumentType: 'ActionArgument',
    }
  };
}

```

```

    argument: faker.argumentForArgumentType(faker.attackingArgumentType()),
},
pathParameters: {
    argumentId: supportedArgument.argumentId,
},
headers: {
    'mocked-user-id': faker.datatype.uuid(),
},
};

await expect(main(event)).rejects.toThrow(
    'argument does not match argumentType'
);
});

it('should add argument id to supportedByArgumentId attribute of parent argument', async () => {
const userTableEntry = faker.userTableEntry();

await putUser(userTableEntry);

const supportedArgument = faker.argumentTableEntry({
    attackedByArgumentIds: [],
    supportedByArgumentIds: []
});

await putArgument(supportedArgument);

const child1ArgumentType = faker.attackingArgumentType();
const child1Event = {
    body: {
        argumentType: child1ArgumentType,
        argument: faker.argumentForArgumentType(child1ArgumentType),
    },
    pathParameters: {
        argumentId: supportedArgument.argumentId,
    },
    headers: {
        'mocked-user-id': userTableEntry.userId,
    },
};

const child1 = await main(child1Event);

```

```

const child1Response = JSON.parse(child1.body);
const child1ResponseMessage = child1Response.message as ArgumentTableEntry;

const child2ArgumentType = faker.attackingArgumentType();
const child2Event = {
  body: {
    argumentType: child2ArgumentType,
    argument: faker.argumentForArgumentType(child2ArgumentType),
  },
  pathParameters: {
    argumentId: supportedArgument.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

const child2 = await main(child2Event);
const child2Response = JSON.parse(child2.body);
const child2ResponseMessage = child2Response.message as ArgumentTableEntry;

const { Item } = await dynamoDb
  .get({
    TableName: ARGUMENT_TABLE_NAME,
    Key: { argumentId: supportedArgument.argumentId },
  })
  .promise();

expect(Item.supportedByArgumentIds).toContain(
  child1ResponseMessage.argumentId
);
expect(Item.supportedByArgumentIds).toContain(
  child2ResponseMessage.argumentId
);
});

});
}
);

```

### E.1.34 argupedia-api-master/src/functions/createNewSupportingArgument/handler.ts

```
import { addArgumentIdForUserAuthoredArgumentIds } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';
import { v4 as uuidv4 } from 'uuid';

import {
  getArgumentById,
  putArgument,
} from '../../../../../adapters/database/argumentTable';
import { labelArguments, validateArgument } from '../../../../../helpers/argupedia';
import {
  Argument,
  ArgumentTableEntry,
  CreateNewArgumentHandlerBody,
} from '../../../../../types/arguments';
import schema from './schema';

const createNewAttackingArgument: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  validateArgument(event.body as CreateNewArgumentHandlerBody);

  if (
    event.pathParameters === undefined ||
    event.pathParameters.argumentId === undefined
```

```

) {

  throw new Error('No\u00a5argumentId\u00a5specified');
}

const supportsArgument = await getArgumentById(
  event.pathParameters.argumentId
);

const createdDate = new Date().toISOString();

const argument = event.body.argument as Argument;

const argumentTableEntry: ArgumentTableEntry = {
  argumentId: uuidv4(),
  argumentType: event.body.argumentType,
  argument: argument,
  argumentText: event.body.argumentText,
  argumentDepth: supportsArgument.argumentDepth,
  authorId: userId,
  createdDate: createdDate,
  modifiedDate: createdDate,
  attacksArgumentId: undefined,
  supportsArgumentId: event.pathParameters.argumentId,
  attackedByArgumentIds: new Array<string>(),
  supportedByArgumentIds: new Array<string>(),
  upvotedByUserIds: new Array<string>(),
  downvotedByUserIds: new Array<string>(),
  label: supportsArgument.label,
  commentIds: new Array<string>(),
};

await putArgument(argumentTableEntry);

await putArgument({
  ...supportsArgument,
  supportedByArgumentIds: supportsArgument.supportedByArgumentIds.concat(
    argumentTableEntry.argumentId
  ),
});

await labelArguments(argumentTableEntry.argumentId);

```

```
await addArgumentIdForUserAuthoredArgumentIds(
  userId,
  argumentTableEntry.argumentId
);

return formatJSONResponse({
  message: argumentTableEntry,
  event,
})};

export const main = middyfy(createNewAttackingArgument);
```

### E.1.35 argupedia-api-master/src/functions/createNewSupportingArgument/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

import schema from './schema';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'post',
        path: 'rest/api/1/argument/{argumentId}/supporting',
        cors: true,
        request: {
          schemas: {
            'application/json': schema,
          },
        },
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.36 argupedia-api-master/src/functions/createNewSupportingArgument/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

### E.1.37 argupedia-api-master/src/functions/createNewSupportingArgument/schema.ts

```
export default {  
  type: 'object',  
  properties: {  
    argumentType: { type: 'string' },  
    argument: { type: 'object' },  
    argumentText: { type: 'string' },  
  },  
  required: ['argumentType', 'argument'],  
} as const;
```

### E.1.38 argupedia-api-master/src/functions/createUser/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { UserTableEntry } from 'src/types/users';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('createUser', () => {
  let main;

  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
  });

  main = (await import('./handler')).main;
});

afterAll(() => {
  jest.resetModules();
});

it('should_add_new_user', async () => {
  const mockedUserId = faker.datatype.uuid();

  const { body } = await main({
    headers: {
      'mocked-user-id': mockedUserId,
    },
  });

  const { message } = JSON.parse(body);
  const entry = message as UserTableEntry;
```

```

expect(entry.userId).toEqual(mockedUserId);
});

it('should add new user with parameters', async () => {
  const mockedUserId = faker.datatype.uuid();

  const event = {
    body: {
      biography: faker.random.words(30),
      displayName: faker.name.findName(),
      profilePictureURL: faker.image.imageUrl(),
    },
    headers: {
      'mocked-user-id': mockedUserId,
    },
  };

  const { body } = await main(event);
  const { message } = JSON.parse(body);
  const entry = message as UserTableEntry;

  expect(entry.biography).toEqual(event.body.biography);
  expect(entry.displayName).toEqual(event.body.displayName);
  expect(entry.profilePictureURL).toEqual(event.body.profilePictureURL);
  expect(entry.userId).toEqual(mockedUserId);
};

});

```

### E.1.39 argupedia-api-master/src/functions/createUser/handler.ts

```
import { putUser } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';
import { UserTableEntry } from 'src/types/users';

import schema from './schema';

const createUser: ValidatedEventAPIGatewayProxyEvent<typeof schema> = async (
  event
) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  const createdDate = new Date().toISOString();

  const loremPicsumUrl = `https://picsum.photos/seed/${userId}/200/300`;

  const userTableEntry: UserTableEntry = {
    authoredArgumentIds: [],
    authoredCommentIds: [],
    userId: userId,
    biography: event.body ? event.body.biography || '' : '',
    displayName: event.body ? event.body.displayName || '' : '',
    lastSignedOn: createdDate,
    profilePictureURL: event.body
      ? event.body.profilePictureURL || loremPicsumUrl
      : loremPicsumUrl,
    votedArgumentIds: []
  };
}
```

```
};

await putUser(userTableEntry);

return formatJSONResponse({
  message: userTableEntry,
  event,
});

};

export const main = middyfy(createUser);
```

### E.1.40 argupedia-api-master/src/functions/createUser/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

import schema from './schema';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'post',
        path: 'rest/api/1/user',
        cors: true,
        request: {
          schemas: {
            'application/json': schema,
          },
        },
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
        documentation: {
          description: 'Create a new user',
          summary: 'Create a new user',
          requestBody: {
            description: 'User object',
          },
          requestModels: {
            'application/json': schema,
          },
          methodResponses: [
            {
              statusCode: 201,
              responseBody: {
                description: 'User object',
              },
              responseModels: {
                'application/json': schema,
              }
            }
          ]
        }
      }
    }
  ]
}
```

```
 },  
 },  
 ],  
 },  
 },  
 ],  
 };
```

#### E.1.41 argupedia-api-master/src/functions/createUser/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.42 argupedia-api-master/src/functions/createUser/schema.ts

```
export default {  
  type: 'object',  
  properties: {  
    biography: { type: 'string' },  
    displayName: { type: 'string' },  
    profilePictureURL: { type: 'string' },  
  },  
} as const;
```

### E.1.43 argupedia-api-master/src/functions/deleteArgumentWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { getArgumentById, putArgument } from '@adapters/database/argumentTable';
import { getUserId, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

jest.mock('@libs/lambda');

describe('deleteArgumentWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeAll(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should_delete_argument', async () => {
    const userTableEntry = faker.userTableEntry();

    await putUser(userTableEntry);

    const mockedArgumentTableEntry = faker.argumentTableEntry({
      authorId: userTableEntry.userId,
    });
  });
});
```

```

await putArgument(mockedArgumentTableEntry);

const { body } = await main({
  pathParameters: {
    argumentId: mockedArgumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
});

const { message } = JSON.parse(body);
const entry = message as ArgumentTableEntry;

expect(entry).toEqual({});

expect(
  await getArgumentById(mockedArgumentTableEntry.argumentId)
).toBeUndefined();

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.authoredArgumentIds).not.toContain(
  mockedArgumentTableEntry.argumentId
);
});

it('should reject any operation where argumentId is not valid', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  await expect(
    main({
      pathParameters: {
        argumentId: 'INVALID-ARGUMENT-ID',
      },
      headers: {
        'mocked-user-id': userTableEntry.userId,
      },
    })
  )
});

```

```

    ).rejects.toThrow('No argument with specified argumentId');

});

it('should reject any delete operation on an argument called by a non-author', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const mockedArgumentTableEntry = faker.argumentTableEntry();

  await putArgument(mockedArgumentTableEntry);

  await expect(
    main({
      pathParameters: {
        argumentId: mockedArgumentTableEntry.argumentId,
      },
      headers: {
        'mocked-user-id': userTableEntry.userId,
      },
    })
  ).rejects.toThrow('User is not the author of the specified argument');
});

it('should delete argument with supporting arguments', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const argumentA = faker.argumentTableEntry({
    authorId: userTableEntry.userId,
  });

  const argumentB = faker.argumentTableEntry({
    authorId: userTableEntry.userId,
    supportsArgumentId: argumentA.argumentId,
  });

  const argumentC = faker.argumentTableEntry({
    authorId: userTableEntry.userId,
    supportsArgumentId: argumentB.argumentId,
  });
});

```

```

argumentA.supportedByArgumentIds = [argumentB.argumentId];
argumentB.supportedByArgumentIds = [argumentC.argumentId];

await putArgument(argumentA);
await putArgument(argumentB);
await putArgument(argumentC);

await main({
  pathParameters: {
    argumentId: argumentB.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
});

const updatedArgumentA = await getArgumentById(argumentA.argumentId);
const updatedArgumentC = await getArgumentById(argumentC.argumentId);

expect(updatedArgumentA.supportedByArgumentIds).toContain(
  argumentC.argumentId
);
expect(updatedArgumentC.supportsArgumentId).toEqual(argumentA.argumentId);
});
});

```

## E.1.44 argupedia-api-master/src/functions/deleteArgumentWithId/handler.ts

```
import { removeArgumentIdForUserAuthoredArgumentIds } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';

import {
  deleteArgumentById,
  getArgumentById,
  putArgument,
} from '../../../../../adapters/database/argumentTable';
import schema from './schema';

const deleteArgumentWithId: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  const argumentId = event.pathParameters.argumentId;

  const retrievedArgument = await getArgumentById(argumentId);

  if (retrievedArgument === undefined) {
    throw new Error(`No argument with specified argumentId`);
  }

  if (retrievedArgument.authorId !== userId) {
    throw new Error(`User is not the author of the specified argument`);
  }
}
```

```

if (retrievedArgument.supportsArgumentId !== undefined) {
  const rootArgument = await getArgumentById(
    retrievedArgument.supportsArgumentId
  );

  rootArgument.supportedByArgumentIds.splice(
    rootArgument.supportedByArgumentIds.indexOf(retrievedArgument.argumentId),
    1
  );
  rootArgument.supportedByArgumentIds.push(
    ...retrievedArgument.supportedByArgumentIds
  );
  await putArgument(rootArgument);

  for (const supportedByArgumentId of retrievedArgument.supportedByArgumentIds) {
    const supportedByArgument = await getArgumentById(supportedByArgumentId);

    supportedByArgument.supportsArgumentId =
      retrievedArgument.supportsArgumentId;
    await putArgument(supportedByArgument);
  }
}

const deleteArgumentResponse = await deleteArgumentById(argumentId);

await removeArgumentIdForUserAuthoredArgumentIds(userId, argumentId);

//TODO - remove all comments associated with argument

return formatJSONResponse({
  message: deleteArgumentResponse,
  event,
});
};

export const main = middyfy(deleteArgumentWithId);

```

### E.1.45 argupedia-api-master/src/functions/deleteArgumentWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'delete',
        path: 'rest/api/1/argument/{argumentId}',
        cors: true,
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.46 argupedia-api-master/src/functions/deleteArgumentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

### E.1.47 argupedia-api-master/src/functions/deleteArgumentWithId/schema.ts

```
export default {} as const;
```

## E.1.48 argupedia-api-master/src/functions/deleteCommentWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { getArgumentById, putArgument } from '@adapters/database/argumentTable';
import { getCommentById, putComment } from '@adapters/database/commentTable';
import { getUserById, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { CommentTableEntry } from 'src/types/comments';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('deleteCommentWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
  });

  main = (await import('./handler')).main;
});

afterAll(() => {
  jest.resetModules();
});

it('should_delete_comment', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const argumentTableEntry = faker.argumentTableEntry();
```

```

const commentTableEntry = faker.commentTableEntry({
  authorId: userTableEntry.userId,
  argumentId: argumentTableEntry.argumentId,
});

argumentTableEntry.commentIds.push(commentTableEntry.commentId);

await putComment(commentTableEntry);
await putArgument(argumentTableEntry);

const { body } = await main({
  pathParameters: {
    commentId: commentTableEntry.commentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
});

const { message } = JSON.parse(body);
const entry = message as CommentTableEntry;

expect(entry).toEqual({});

expect(await getCommentById(commentTableEntry.commentId)).toBeUndefined();

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.authoredCommentIds).not.toContain(
  commentTableEntry.commentId
);

const updatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(updatedArgumentTableEntry.commentIds).not.toContain(
  commentTableEntry.commentId
);
});

it('should reject any operation where argumentId is not valid', async () => {

```

```

const userTableEntry = faker.userTableEntry();

await putUser(userTableEntry);

await expect(
  main({
    pathParameters: {
      commentId: 'INVALID-COMMENT-ID',
    },
    headers: {
      'mocked-user-id': userTableEntry.userId,
    },
  })
).rejects.toThrow('No comment with specified commentId');

});

it('should reject any delete operation on an argument called by a non-author', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const commentTableEntry = faker.commentTableEntry();

  await putComment(commentTableEntry);

  await expect(
    main({
      pathParameters: {
        commentId: commentTableEntry.commentId,
      },
      headers: {
        'mocked-user-id': userTableEntry.userId,
      },
    },
  })
).rejects.toThrow('User is not the author of the specified comment');

});
});

```

## E.1.49 argupedia-api-master/src/functions/deleteCommentWithId/handler.ts

```
import { removeCommentIdFromArgumentCommentIds } from '@adapters/database/argumentTable';
import {
  deleteCommentById,
  getCommentById,
} from '@adapters/database/commentTable';

import { removeCommentIdForUserAuthoredCommentIds } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';

import schema from './schema';

const deleteCommentWithId: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  const commentId = event.pathParameters.commentId;

  const retrievedComment = await getCommentById(commentId);

  if (retrievedComment === undefined) {
    throw new Error('No comment with specified commentId');
  }

  if (retrievedComment.authorId !== userId) {
    throw new Error('User is not the author of the specified comment');
  }
}
```

```
const deleteCommentResponse = await deleteCommentById(commentId);

await removeCommentIdForUserAuthoredCommentIds(userId, commentId);
await removeCommentIdFromArgumentCommentIds(
  retrievedComment.argumentId,
  commentId
);

return formatJSONResponse({
  message: deleteCommentResponse,
  event,
});
};

export const main = middyfy(deleteCommentWithId);
```

### E.1.50 argupedia-api-master/src/functions/deleteCommentWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'delete',
        path: 'rest/api/1/comment/{commentId}',
        cors: true,
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.51 argupedia-api-master/src/functions/deleteCommentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.52 argupedia-api-master/src/functions/deleteCommentWithId/schema.ts

```
export default {} as const;
```

### E.1.53 argupedia-api-master/src/functions/deleteUserWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { getArgumentById, putArgument } from '@adapters/database/argumentTable';
import { getCommentById, putComment } from '@adapters/database/commentTable';
import { getUserId, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { UserTableEntry } from 'src/types/users';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('deleteUserWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
  });

  main = (await import('./handler')).main;
});

afterAll(() => {
  jest.resetModules();
});

it('should_delete_user', async () => {
  const userTableEntry = faker.userTableEntry();

  const argumentTableEntry = faker.argumentTableEntry({
    authorId: userTableEntry.userId,
    upvotedByUserIds: [userTableEntry.userId],
  });

```

```

const commentTableEntry = faker.commentTableEntry({
  authorId: userTableEntry.userId,
  argumentId: argumentTableEntry.argumentId,
});

userTableEntry.authoredArgumentIds = [argumentTableEntry.argumentId];
userTableEntry.authoredCommentIds = [commentTableEntry.commentId];
userTableEntry.votedArgumentIds = [argumentTableEntry.argumentId];

argumentTableEntry.commentIds = [commentTableEntry.commentId];

await putUser(userTableEntry);
await putArgument(argumentTableEntry);
await putComment(commentTableEntry);

const { body } = await main({
  pathParameters: {
    userId: userTableEntry.userId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
});

const { message } = JSON.parse(body);
const entry = message as UserTableEntry;

expect(entry).toEqual({});

expect(await getUserById(userTableEntry.userId)).toBeUndefined();
expect(
  await getArgumentById(argumentTableEntry.argumentId)
).not.toBeUndefined();
expect(await getCommentById(commentTableEntry.commentId)).toBeUndefined();
});

it('should reject any operation where userId is not valid', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);
}

```

```

const event = {
  body: {
    biography: faker.random.words(30),
    displayName: faker.name.findName(),
    profilePictureURL: faker.image.imageUrl(),
  },
  pathParameters: {
    userId: faker.datatype.uuid(),
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await expect(main(event)).rejects.toThrow('No user with specified userId');
});

it('should reject any operation where userId does not match logged in user', async () => {
  const user1TableEntry = faker.userTableEntry();
  const user2TableEntry = faker.userTableEntry();

  await putUser(user1TableEntry);
  await putUser(user2TableEntry);

  const event = {
    body: {
      biography: faker.random.words(30),
      displayName: faker.name.findName(),
      profilePictureURL: faker.image.imageUrl(),
    },
    pathParameters: {
      userId: user1TableEntry.userId,
    },
    headers: {
      'mocked-user-id': user2TableEntry.userId,
    },
  };

  await expect(main(event)).rejects.toThrow(
    'User cannot update a different user'
  );
});

```

} ;

### E.1.54 argupedia-api-master/src/functions/deleteUserWithId/handler.ts

```
import {
  removeCommentIdFromArgumentCommentIds,
  removeUserIdFromArgumentDownvotes,
  removeUserIdFromArgumentUpvotes,
} from '@adapters/database/argumentTable';

import {
  deleteCommentById,
  getCommentById,
} from '@adapters/database/commentTable';

import { deleteUserById, getUserById } from '@adapters/database/userTable';
import { deleteCognitoUserWithId } from '@helpers/Cognito';
import { verifyJwtToken } from '@helpers/JWT';

import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';

import schema from './schema';

const deleteUserWithId: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  const retrievedUser = await getUserById(event.pathParameters.userId);

  if (retrievedUser === undefined) {
    throw new Error('No user with specified userId');
  }
}
```

```

if (retrievedUser.userId !== userId) {
  throw new Error('User cannot update a different user');
}

for (const argumentId of retrievedUser.votedArgumentIds) {
  await removeUserIdFromArgumentDownvotes(argumentId, userId);
  await removeUserIdFromArgumentUpvotes(argumentId, userId);
}

for (const commentId of retrievedUser.authoredCommentIds) {
  const comment = await getCommentById(commentId);

  await removeCommentIdFromArgumentCommentIds(comment.argumentId, commentId);
  await deleteCommentById(commentId);
}

//Do not delete arguments, instead do what reddit does and show user as [Deleted]

const deleteUserResponse = await deleteUserById(userId);

if (!isTest) {
  await deleteCognitoUserWithId(userId);
}

return formatJSONResponse({
  message: deleteUserResponse,
  event,
});
};

export const main = middyfy(deleteUserWithId);

```

### E.1.55 argupedia-api-master/src/functions/deleteUserWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'delete',
        path: 'rest/api/1/user/{userId}',
        cors: true,
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.56 argupedia-api-master/src/functions/deleteUserWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.57 argupedia-api-master/src/functions/deleteUserWithId/schema.ts

```
export default {} as const;
```

## E.1.58 argupedia-api-master/src/functions/downvoteArgumentWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import {
    addUserToArgumentUpvotes,
    getArgumentById,
    putArgument,
} from '@adapters/database/argumentTable';

import { getUserById, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('downvoteArgumentWithId', () => {
    let main;
    let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

    beforeEach(async () => {
        mockedMiddyfy = mocked(middyfy);
        mockedMiddyfy.mockImplementation((handler: Handler) => {
            return handler as never;
        });

        main = (await import('./handler')).main;
    });

    afterAll(() => {
        jest.resetModules();
    });
}

it('should_add_user_id_to_argument_downvotes', async () => {
    const userTableEntry = faker.userTableEntry();

    await putUser(userTableEntry);

    expect(await addUserToArgumentUpvotes(userTableEntry.id)).toEqual(true);
});
```

```

const argumentTableEntry = faker.argumentTableEntry();

await putArgument(argumentTableEntry);

expect(argumentTableEntry.downvotedByUserIds).not.toContain(
  userTableEntry.userId
);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await main(event);

const updatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(updatedArgumentTableEntry.downvotedByUserIds).toContain(
  userTableEntry.userId
);

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.votedArgumentIds).toContain(
  argumentTableEntry.argumentId
);
});

it('should not add user id twice to downvotes', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const argumentTableEntry = faker.argumentTableEntry();

  await putArgument(argumentTableEntry);
}

```

```

expect(argumentTableEntry.downvotedByUserIds).not.toContain(
  userTableEntry.userId
);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await main(event);
await main(event);

const updatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

const downvoteSet = new Set(updatedArgumentTableEntry.downvotedByUserIds);

expect(updatedArgumentTableEntry.downvotedByUserIds.length).toEqual(
  downvoteSet.size
);
});

it('should_remove_user_id_from_downvotes_when_adding_to_upvotes', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const argumentTableEntry = faker.argumentTableEntry();

  await putArgument(argumentTableEntry);

  expect(argumentTableEntry.upvotedByUserIds).not.toContain(
    userTableEntry.userId
);
  expect(argumentTableEntry.downvotedByUserIds).not.toContain(
    userTableEntry.userId
);
}

```

```

);

await addUserIdToArgumentUpvotes(
  argumentTableEntry.argumentId,
  userTableEntry.userId
);

const middleUpdatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(middleUpdatedArgumentTableEntry.upvotedByUserIds).toContain(
  userTableEntry.userId
);
expect(middleUpdatedArgumentTableEntry.downvotedByUserIds).not.toContain(
  userTableEntry.userId
);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await main(event);

const finalUpdatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(finalUpdatedArgumentTableEntry.upvotedByUserIds).not.toContain(
  userTableEntry.userId
);
expect(finalUpdatedArgumentTableEntry.downvotedByUserIds).toContain(
  userTableEntry.userId
);
});

it('should reject any operation where argumentId is not valid', async () => {

```

```
const userTableEntry = faker.userTableEntry();

await putUser(userTableEntry);

const event = {
  pathParameters: {
    argumentId: faker.datatype.uuid(),
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await expect(main(event)).rejects.toThrow(
  'No argument with specified argumentId'
);
});

});
```

## E.1.59 argupedia-api-master/src/functions/downvoteArgumentWithId/handler.ts

```
import { addArgumentIdForUserVotedArgumentIds } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';

import {
  addUserIdToArgumentDownvotes,
  getArgumentById,
  removeUserIdFromArgumentUpvotes,
} from '../../adapters/database/argumentTable';
import schema from './schema';

const downvoteArgumentWithId: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  const argumentId = event.pathParameters.argumentId;

  const argumentTableEntry = await getArgumentById(argumentId);

  if (argumentTableEntry === undefined) {
    throw new Error('No argument with specified argumentId');
  }

  await removeUserIdFromArgumentUpvotes(argumentId, userId);
  await addUserIdToArgumentDownvotes(argumentId, userId);
  await addArgumentIdForUserVotedArgumentIds(userId, argumentId);
}
```

```
const argumentToBeReturned = await getArgumentById(argumentId);

return formatJSONResponse({
  message: argumentToBeReturned,
  event,
});

export const main = middyfy(downvoteArgumentWithId);
```

## E.1.60 argupedia-api-master/src/functions/downvoteArgumentWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'post',
        path: 'rest/api/1/argument/{argumentId}/downvote',
        cors: true,
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    ],
  };
};
```

### E.1.61 argupedia-api-master/src/functions/downvoteArgumentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

## E.1.62 argupedia-api-master/src/functions/downvoteArgumentWithId/schema.ts

```
export default {} as const;
```

### E.1.63 argupedia-api-master/src/functions/getAllArguments/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putArgument } from '@adapters/database/argumentTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { ArgumentTableEntry } from 'src/types/arguments';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('getAllArguments', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should_get_all_arguments', async () => {
    const tableEntries = [
      faker.argumentTableEntry(),
      faker.argumentTableEntry(),
      faker.argumentTableEntry(),
      faker.argumentTableEntry(),
      faker.argumentTableEntry(),
    ];

    for (const argument of tableEntries) {
```

```
    await putArgument(argument);
}

const { body } = await main({});

const { message } = JSON.parse(body);
const retrievedEntries = message.Items as ArgumentTableEntry[];

const retrievedIds = retrievedEntries.map((entry) => entry.argumentId);

for (const argument of tableEntries) {
  expect(retrievedIds).toContain(argument.argumentId);
}

});

});
```

## E.1.64 argupedia-api-master/src/functions/getAllArguments/handler.ts

```
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';

import { getArguments } from '../../../../../adapters/database/argumentTable';
import schema from './schema';

const getAllArguments: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    const allArguments = await getArguments();

    return formatJSONResponse({
        message: allArguments,
        event,
    });
};

export const main = middyfy(getAllArguments);
```

### E.1.65 argupedia-api-master/src/functions/getAllArguments/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'get',
        path: 'rest/api/1/argument/',
        cors: true,
      },
    },
  ],
};

};
```

### E.1.66 argupedia-api-master/src/functions/getAllArguments/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.67 argupedia-api-master/src/functions/getAllArguments/schema.ts

```
export default {} as const;
```

## E.1.68 argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putArgument } from '@adapters/database/argumentTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

jest.mock('@libs/lambda');

describe('getArgumentGraphForArgumentWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  it('should get all arguments in an argument tree', async () => {
    const argumentA = faker.argumentTableEntry({
      supportedByArgumentIds: [],
      attackedByArgumentIds: [],
    });
    const argumentB = faker.argumentTableEntry({
      supportedByArgumentIds: [],
      attackedByArgumentIds: [],
      attacksArgumentId: argumentA.argumentId,
    });
  });
});
```

```

});

const argumentC = faker.argumentTableEntry({
  supportedByArgumentIds: [],
  attackedByArgumentIds: [],
  attacksArgumentId: argumentA.argumentId,
});

const argumentD = faker.argumentTableEntry({
  supportedByArgumentIds: [],
  attackedByArgumentIds: [],
  supportsArgumentId: argumentC.argumentId,
});

const argumentE = faker.argumentTableEntry({
  supportedByArgumentIds: [],
  attackedByArgumentIds: [],
  attacksArgumentId: argumentC.argumentId,
});

argumentA.attackedByArgumentIds = [
  argumentB.argumentId,
  argumentC.argumentId,
];
argumentC.attackedByArgumentIds = [argumentE.argumentId];
argumentC.supportedByArgumentIds = [argumentD.argumentId];

await putArgument(argumentA);
await putArgument(argumentB);
await putArgument(argumentC);
await putArgument(argumentD);
await putArgument(argumentE);

const event = {
  pathParameters: {
    argumentId: argumentC.argumentId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const retrievedEntry = message as Array<ArgumentTableEntry>

const idsPresent = retrievedEntry.map((entry) => entry.argumentId);

```

```

expect(idsPresent).toContain(argumentA.argumentId);
expect(idsPresent).toContain(argumentB.argumentId);
expect(idsPresent).toContain(argumentC.argumentId);
expect(idsPresent).toContain(argumentD.argumentId);
expect(idsPresent).toContain(argumentE.argumentId);

});

it('should_reject_any_operation_where_argumentId_does_not_exist', async () => {
  const event = {
    pathParameters: {
      argumentId: faker.datatype.uuid(),
    },
  };

  await expect(main(event)).rejects.toThrow(
    'No_argument_with_specified_argumentId'
  );
});
});
});

```

## E.1.69 argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/

```
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { ArgumentTableEntry } from 'src/types/arguments';

import { getArgumentById } from '../../adapters/database/argumentTable';
import schema from './schema';

const getRootArgumentId = async (argumentId: string): Promise<string> => {
    const argument = await getArgumentById(argumentId);

    if (argument.supportsArgumentId !== undefined) {
        return getRootArgumentId(argument.supportsArgumentId);
    } else if (argument.attacksArgumentId !== undefined) {
        return getRootArgumentId(argument.attacksArgumentId);
    } else {
        return argument.argumentId;
    }
};

const getAllArgumentsInTreeRecursive = async (
    argumentId: string
): Promise<Array<ArgumentTableEntry>> => {
    const argument = await getArgumentById(argumentId);
    const argumentList = [argument];

    for (const attackedByArgumentId of argument.attackedByArgumentIds) {
        argumentList.push(
            ...await getAllArgumentsInTreeRecursive(attackedByArgumentId)
        );
    }

    for (const supportedByArgumentId of argument.supportedByArgumentIds) {
        argumentList.push(
            ...await getAllArgumentsInTreeRecursive(supportedByArgumentId)
        );
    }
};

return argumentList;
};
```

```

const getArgumentGraphForArgumentWithId: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  const argumentId = event.pathParameters.argumentId;

  const argumentTableEntry = await getArgumentById(argumentId);

  if (argumentTableEntry === undefined) {
    throw new Error('No argument with specified argumentId');
  }

  const rootArgumentId = await getRootArgumentId(argumentId);

  const output = await getAllArgumentsInTreeRecursive(rootArgumentId);

  return formatJSONResponse({
    message: output,
    event,
  });
};

export const main = middyfy(getArgumentGraphForArgumentWithId);

```

### E.1.70 argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/index.js

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'get',
        path: 'rest/api/1/argument/{argumentId}/graph',
        cors: true,
      },
    },
  ],
};

};
```

### E.1.71 argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/n

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

### E.1.72 argupedia-api-master/src/functions/getArgumentGraphForArgumentWithId/s

```
export default {} as const;
```

### E.1.73 argupedia-api-master/src/functions/getArgumentWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putArgument } from '@adapters/database/argumentTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

jest.mock('@libs/lambda');

describe('getArgumentWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  it('should get argument by ID', async () => {
    const argument = faker.argumentTableEntry();

    await putArgument(argument);

    const event = {
      pathParameters: {
        argumentId: argument.argumentId,
      },
    };

    const response = await main(event);
    expect(response).toEqual(argument);
  });
});
```

```

};

const { body } = await main(event);
const { message } = JSON.parse(body);
const retrievedEntry = message as ArgumentTableEntry;

expect(argument).toEqual(retrievedEntry);
});

it('should reject any operation where argumentId does not exist', async () => {
const event = {
pathParameters: {
argumentId: faker.datatype.uuid(),
},
};

await expect(main(event)).rejects.toThrow(
'No argument with specified argumentId'
);
});

});
});

```

### E.1.74 argupedia-api-master/src/functions/getArgumentWithId/handler.ts

```
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';

import { getArgumentById } from '../../adapters/database/argumentTable';
import schema from './schema';

const getArgumentWithId: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    const argumentId = event.pathParameters.argumentId;

    const argumentTableEntry = await getArgumentById(argumentId);

    if (argumentTableEntry === undefined) {
        throw new Error('No argument with specified argumentId');
    }

    return formatJSONResponse({
        message: argumentTableEntry,
        event,
    });
};

export const main = middyfy(getArgumentWithId);
```

### E.1.75 argupedia-api-master/src/functions/getArgumentWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'get',
        path: 'rest/api/1/argument/{argumentId}',
        cors: true,
      },
    },
  ],
};

};
```

### E.1.76 argupedia-api-master/src/functions/getArgumentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.77 argupedia-api-master/src/functions/getArgumentWithId/schema.ts

```
export default {} as const;
```

## E.1.78 argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/h

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putArgument } from '@adapters/database/argumentTable';
import { putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

jest.mock('@libs/lambda');

describe('getAuthoredArgumentsForUserWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeAll(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should get arguments authored by user with userId', async () => {
    const userTableEntry = faker.userTableEntry();

    const argumentList = new Array(10)
      .fill(null)
      .map(() => faker.argumentTableEntry({ authorId: userTableEntry.userId }));

    for (const argumentTableEntry of argumentList) {
```

```

    await putArgument(argumentTableEntry);
}

userTableEntry.authoredArgumentIds = argumentList.map(
  (argument) => argument.argumentId
);

await putUser(userTableEntry);

const event = {
  pathParameters: {
    userId: userTableEntry.userId,
  },
  queryStringParameters: {
    count: `${faker.datatype.number({ min: 1, max: 10 })}`,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const retrievedEntry = message as ArgumentTableEntry[];

argumentList.sort((a, b) => b.createdDate.localeCompare(a.createdDate));

const numItems =
  event.queryStringParameters.count === undefined
  ? argumentList.length
  : Math.min(
    argumentList.length,
    parseInt(event.queryStringParameters.count)
  );
const smallList = argumentList.slice(0, numItems);

expect(smallList).toEqual(retrievedEntry);
});

it('should reject any operation where userId does not exist', async () => {
  const event = {
    pathParameters: {
      userId: faker.datatype.uuid(),
    },
  };
}

```

```
    await expect(main(event)).rejects.toThrow('No user with specified userId');
  });
});
```

### E.1.79 argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/h

```
import { getUserId } from '@adapters/database/userTable';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';

import { getArgumentById } from '../../adapters/database/argumentTable';
import schema from './schema';

const getAuthoredArgumentsForUserWithId: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    const userId = event.pathParameters.userId;

    const userTableEntry = await getUserId(userId);

    if (userTableEntry === undefined) {
        throw new Error('No user with specified userId');
    }

    const argumentsList = await Promise.all(
        userTableEntry.authoredArgumentIds.map(
            async (argumentId) => await getArgumentById(argumentId)
        )
    );
}

argumentsList.sort((a, b) => b.createdDate.localeCompare(a.createdDate));

const numItems =
    event.queryStringParameters.count === undefined
    ? argumentsList.length
    : Math.min(
        argumentsList.length,
        parseInt(event.queryStringParameters.count)
    );
const smallList = argumentsList.slice(0, numItems);

return formatJSONResponse({
    message: smallList,
    event,
});
```

};

```
export const main = middyfy(getAuthoredArgumentsForUserWithId);
```

## E.1.80 argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/in

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'get',
        path: 'rest/api/1/user/{userId}/authoredArguments',
        cors: true,
      },
    },
  ],
};

};
```

### E.1.81 argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/m

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

### E.1.82 argupedia-api-master/src/functions/getAuthoredArgumentsForUserWithId/sc

```
export default {} as const;
```

### E.1.83 argupedia-api-master/src/functions/getAuthoredCommentsForUserWithId/handler

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putComment } from '@adapters/database/commentTable';
import { putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

jest.mock('@libs/lambda');

describe('getAuthoredArgumentsForUserWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeAll(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should get comments authored by user with userId', async () => {
    const userTableEntry = faker.userTableEntry();

    const commentList = new Array(10)
      .fill(null)
      .map(() => faker.commentTableEntry({ authorId: userTableEntry.userId }));

    for (const commentTableEntry of commentList) {
```

```

    await putComment(commentTableEntry);
}

userTableEntry.authoredCommentIds = commentList.map(
  (comment) => comment.commentId
);

await putUser(userTableEntry);

const event = {
  pathParameters: {
    userId: userTableEntry.userId,
  },
  queryStringParameters: {
    count: `${faker.datatype.number({ min: 1, max: 10 })}`,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const retrievedEntry = message as ArgumentTableEntry[];

commentList.sort((a, b) => b.createdDate.localeCompare(a.createdDate));

const numItems =
  event.queryStringParameters.count === undefined
    ? commentList.length
    : Math.min(
        commentList.length,
        parseInt(event.queryStringParameters.count)
      );
const smallList = commentList.slice(0, numItems);

expect(smallList).toEqual(retrievedEntry);
});

it('should reject any operation where userId does not exist', async () => {
  const event = {
    pathParameters: {
      userId: faker.datatype.uuid(),
    },
  };
}

```

```
    await expect(main(event)).rejects.toThrow('No user with specified userId');
  });
});
```

## E.1.84 argupedia-api-master/src/functions/getAuthoredCommentsForUserWithId/handler

```
import { getCommentById } from '@adapters/database/commentTable';
import { getUserId } from '@adapters/database/userTable';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';

import schema from './schema';

const getAuthoredCommentsForUserWithId: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    const userId = event.pathParameters.userId;

    const userTableEntry = await getUserId(userId);

    if (userTableEntry === undefined) {
        throw new Error('No user with specified userId');
    }

    const commentsList = await Promise.all(
        userTableEntry.authoredCommentIds.map(
            async (commentId) => await getCommentById(commentId)
        )
    );
}

commentsList.sort((a, b) => b.createdDate.localeCompare(a.createdDate));

const numItems =
    event.queryStringParameters.count === undefined
    ? commentsList.length
    : Math.min(
        commentsList.length,
        parseInt(event.queryStringParameters.count)
    );
const smallList = commentsList.slice(0, numItems);

return formatJSONResponse({
    message: smallList,
    event,
});
```

};

```
export const main = middyfy(getAuthoredCommentsForUserWithId);
```

### E.1.85 argupedia-api-master/src/functions/getAuthoredCommentsForUserWithId/index.js

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'get',
        path: 'rest/api/1/user/{userId}/authoredComments',
        cors: true,
      },
    },
  ],
};

};
```

### E.1.86 argupedia-api-master/src/functions/getAuthoredCommentsForUserId/main.js

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

### E.1.87 argupedia-api-master/src/functions/getAuthoredCommentsForUserId/sch

```
export default {} as const;
```

## E.1.88 argupedia-api-master/src/functions/getCommentWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putArgument } from '@adapters/database/argumentTable';
import { putComment } from '@adapters/database/commentTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { CommentTableEntry } from 'src/types/comments';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('getCommentWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeAll(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should get comment with id', async () => {
    const commentTableEntry = faker.commentTableEntry();

    const argumentTableEntry = faker.argumentTableEntry({
      commentIds: [commentTableEntry.commentId],
    });

    await putComment(commentTableEntry);

    const result = await main(argumentTableEntry);
    expect(result).toEqual(commentTableEntry);
  });
});
```

```

await putArgument(argumentTableEntry);

const event = {
  pathParameters: {
    commentId: commentTableEntry.commentId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as CommentTableEntry;

expect(entry.commentBody).toEqual(commentTableEntry.commentBody);
expect(entry.commentId).toEqual(commentTableEntry.commentId);
});

it('should reject any operation where commentId is not valid', async () => {
  const event = {
    pathParameters: {
      commentId: faker.datatype.uuid(),
    },
  };

  await expect(main(event)).rejects.toThrow(
    'No comment with specified argumentId'
  );
});
});
}

```

### E.1.89 argupedia-api-master/src/functions/getCommentWithId/handler.ts

```
import { getCommentById } from '@adapters/database/commentTable';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';

import schema from './schema';

const getCommentWithId: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    const comment = await getCommentById(event.pathParameters.commentId);

    if (comment === undefined) {
        throw new Error('No comment with specified argumentId');
    }

    return formatJSONResponse({
        message: comment,
        event,
    });
};

export const main = middyfy(getCommentWithId);
```

### E.1.90 argupedia-api-master/src/functions/getCommentWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'get',
        path: 'rest/api/1/comment/{commentId}',
        cors: true,
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.91 argupedia-api-master/src/functions/getCommentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.92 argupedia-api-master/src/functions/getCommentWithId/schema.ts

```
export default {} as const;
```

### E.1.93 argupedia-api-master/src/functions/getCommentsForArgumentWithId/handle

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putArgument } from '@adapters/database/argumentTable';
import { putComment } from '@adapters/database/commentTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { CommentTableEntry } from 'src/types/comments';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('getCommentsForArgumentWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeAll(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should get comment for argument', async () => {
    const commentTableEntry = faker.commentTableEntry();

    const argumentTableEntry = faker.argumentTableEntry({
      commentIds: [commentTableEntry.commentId],
    });

    await putComment(commentTableEntry);

    const result = await main(argumentTableEntry);
    expect(result).toEqual([
      {
        id: argumentTableEntry.id,
        commentId: commentTableEntry.commentId,
        content: commentTableEntry.content,
        author: commentTableEntry.author,
        timestamp: commentTableEntry.timestamp,
      },
    ]);
  });
});
```

```

await putArgument(argumentTableEntry);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as CommentTableEntry[];

expect(entry[0].commentBody).toEqual(commentTableEntry.commentBody);
});

it('should get comments for argument sorted by created date', async () => {
  const commentTableEntry1 = faker.commentTableEntry({
    createdDate: new Date().toISOString(),
  });

  await new Promise((r) => setTimeout(r, 300));

  const commentTableEntry2 = faker.commentTableEntry({
    createdDate: new Date().toISOString(),
  });

  await new Promise((r) => setTimeout(r, 300));

  const commentTableEntry3 = faker.commentTableEntry({
    createdDate: new Date().toISOString(),
  });

  const argumentTableEntry = faker.argumentTableEntry({
    commentIds: [
      commentTableEntry1.commentId,
      commentTableEntry2.commentId,
      commentTableEntry3.commentId,
    ],
  });

  await putComment(commentTableEntry1);
}

```

```

await putComment(commentTableEntry2);
await putComment(commentTableEntry3);

await putArgument(argumentTableEntry);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as CommentTableEntry[];

expect(entry.length).toEqual(3);
expect(entry.map((e) => e.commentId)).toEqual([
  commentTableEntry1.commentId,
  commentTableEntry2.commentId,
  commentTableEntry3.commentId,
]);
});

it('should_reject_any_operation_where_argumentId_is_not_valid', async () => {
  const event = {
    pathParameters: {
      argumentId: faker.datatype.uuid(),
    },
  };

  await expect(main(event)).rejects.toThrow(
    'No argument with specified argumentId'
  );
});
});
});

```

## E.1.94 argupedia-api-master/src/functions/getCommentsForArgumentWithId/handle

```
import { getCommentById } from '@adapters/database/commentTable';
import { getUserId } from '@adapters/database/userTable';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';

import { getArgumentById } from '../../../../../adapters/database/argumentTable';
import schema from './schema';

const getCommentsForArgumentWithId: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    const originalArgument = await getArgumentById(
        event.pathParameters.argumentId
    );

    if (originalArgument === undefined) {
        throw new Error('No argument with specified argumentId');
    }

    const comments = await Promise.all(
        originalArgument.commentIds.map(async (commentId) => {
            const comment = await getCommentById(commentId);
            const user = await getUserId(comment.authorId);

            return {
                ...comment,
                authorDetails: user,
            };
        })
    );

    comments.sort((a, b) => a.createdDate.localeCompare(b.createdDate));

    return formatJSONResponse({
        message: comments,
        event,
    });
};
```

```
export const main = middyfy(getCommentsForArgumentWithId);
```

### E.1.95 argupedia-api-master/src/functions/getCommentsForArgumentWithId/index.js

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'get',
        path: 'rest/api/1/argument/{argumentId}/comments',
        cors: true,
      },
    },
  ],
};

};
```

### E.1.96 argupedia-api-master/src/functions/getCommentsForArgumentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.97 argupedia-api-master/src/functions/getCommentsForArgumentWithId/schema

```
export default {} as const;
```

## E.1.98 argupedia-api-master/src/functions/getUserWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { UserTableEntry } from 'src/types/users';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('getUserWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  it('should get user by ID', async () => {
    const user = faker.userTableEntry();

    await putUser(user);

    const event = {
      pathParameters: {
        userId: user.userId,
      },
    };
  });
});
```

```
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const retrievedEntry = message as UserTableEntry;

expect(user).toEqual(retrievedEntry);
});

it('should reject any operation where userId does not exist', async () => {
const event = {
pathParameters: {
userId: faker.datatype.uuid(),
},
};
await expect(main(event)).rejects.toThrow('No user with specified userId');
});
});
```

### E.1.99 argupedia-api-master/src/functions/getUserWithId/handler.ts

```
import { getUserById } from '@adapters/database/userTable';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';

import schema from './schema';

const getUserWithId: ValidatedEventAPIGatewayProxyEvent<typeof schema> = async (
  event
) => {
  const userId = event.pathParameters.userId;

  const userTableEntry = await getUserById(userId);

  if (userTableEntry === undefined) {
    throw new Error('No user with specified userId');
  }

  return formatJSONResponse({
    message: userTableEntry,
    event,
  });
};

export const main = middyfy(getUserWithId);
```

### E.1.100 argupedia-api-master/src/functions/getUserWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'get',
        path: 'rest/api/1/user/{userId}',
        cors: true,
      },
    },
  ],
};

};
```

### E.1.101 argupedia-api-master/src/functions/getUserWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.102 argupedia-api-master/src/functions/getUserWithId/schema.ts

```
export default {} as const;
```

### E.1.103 argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/hand

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putArgument } from '@adapters/database/argumentTable';
import { putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

jest.mock('@libs/lambda');

describe('getAuthoredArgumentsForUserWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeAll(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should get arguments voted on by user with userId', async () => {
    const userTableEntry = faker.userTableEntry();

    const argumentList = new Array(10)
      .fill(null)
      .map(() => faker.argumentTableEntry({ authorId: userTableEntry.userId }));

    for (const argumentTableEntry of argumentList) {
```

```

    await putArgument(argumentTableEntry);
}

userTableEntry.votedArgumentIds = argumentList.map(
  (argument) => argument.argumentId
);

await putUser(userTableEntry);

const event = {
  pathParameters: {
    userId: userTableEntry.userId,
  },
  queryStringParameters: {
    count: `${faker.datatype.number({ min: 1, max: 10 })}`,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const retrievedEntry = message as ArgumentTableEntry[];

argumentList.sort((a, b) => b.createdDate.localeCompare(a.createdDate));

const numItems =
  event.queryStringParameters.count === undefined
  ? argumentList.length
  : Math.min(
    argumentList.length,
    parseInt(event.queryStringParameters.count)
  );
const smallList = argumentList.slice(0, numItems);

expect(smallList).toEqual(retrievedEntry);
});

it('should reject any operation where userId does not exist', async () => {
  const event = {
    pathParameters: {
      userId: faker.datatype.uuid(),
    },
  };
}

```

```
    await expect(main(event)).rejects.toThrow('No user with specified userId');
  });
});
```

## E.1.104 argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/handler

```
import { getUserId } from '@adapters/database/userTable';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';

import { getArgumentById } from '../../adapters/database/argumentTable';
import schema from './schema';

const getVotedArgumentsForUserWithId: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    const userId = event.pathParameters.userId;

    const userTableEntry = await getUserId(userId);

    if (userTableEntry === undefined) {
        throw new Error('No user with specified userId');
    }

    const argumentsList = await Promise.all(
        userTableEntry.votedArgumentIds.map(
            async (argumentId) => await getArgumentById(argumentId)
        )
    );
}

argumentsList.sort((a, b) => b.createdDate.localeCompare(a.createdDate));

const numItems =
    event.queryStringParameters.count === undefined
    ? argumentsList.length
    : Math.min(
        argumentsList.length,
        parseInt(event.queryStringParameters.count)
    );
const smallList = argumentsList.slice(0, numItems);

return formatJSONResponse({
    message: smallList,
    event,
});
```

};

```
export const main = middyfy(getVotedArgumentsForUserWithId);
```

### E.1.105 argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/index.js

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'get',
        path: 'rest/api/1/user/{userId}/votedArguments',
        cors: true,
      },
    },
  ],
};

};
```

### E.1.106 argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/mock

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

### E.1.107 argupedia-api-master/src/functions/getVotedArgumentsForUserWithId/sche

```
export default {} as const;
```

### E.1.108 argupedia-api-master/src/functions/index.ts

```
export { default as hello } from './hello';
```

## E.1.109 argupedia-api-master/src/functions/updateArgumentWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putArgument } from '@adapters/database/argumentTable';
import { putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';
import { ArgumentTableEntry } from '../../types/arguments';

jest.mock('@libs/lambda');

describe('updateArgumentWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeAll(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should_update_argument', async () => {
    const userTableEntry = faker.userTableEntry();

    await putUser(userTableEntry);

    const originalArgument = faker.argumentTableEntry({
      argumentType: 'ActionArgument',
      argument: faker.argument(),
    });
  });
});
```

```

authorId: userTableEntry.userId,
});

await putArgument(originalArgument);

const event = {
  body: {
    argumentType: 'ActionArgument',
    argument: faker.actionArgument(),
  },
  pathParameters: {
    argumentId: originalArgument.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as ArgumentTableEntry;

expect(entry.argument).toEqual(event.body.argument);
expect(entry.argumentType).toEqual(event.body.argumentType);
});

it('should_update_attacking_argument', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const originalArgumentType = faker.attackingArgumentType();

  const originalArgument = faker.argumentTableEntry({
    argumentType: originalArgumentType,
    argument: faker.argumentForArgumentType(originalArgumentType),
    authorId: userTableEntry.userId,
  });

  await putArgument(originalArgument);

  const newArgumentType = faker.argumentType();

```

```

const event = {
  body: {
    argumentType: newArgumentType,
    argument: faker.argumentForArgumentType(newArgumentType),
  },
  pathParameters: {
    argumentId: originalArgument.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as ArgumentTableEntry;

expect(entry.argument).toEqual(event.body.argument);
expect(entry.argumentType).toEqual(event.body.argumentType);
});

it('should reject any operation where argumentType is not valid', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const originalArgument = faker.argumentTableEntry({
    authorId: userTableEntry.userId,
  });

  await putArgument(originalArgument);

  const event = {
    body: {
      argumentType: 'InvalidArgumentType',
      argument: faker.argument(),
    },
    pathParameters: {
      argumentId: originalArgument.argumentId,
    },
    headers: {

```

```

'mocked-user-id': userTableEntry.userId,
},
};

await expect(main(event)).rejects.toThrow('InvalidArgumentType');
});

it('should_reject_any_operation_where_argumentType_does_not_match_provided_argument', async () => {
const userTableEntry = faker.userTableEntry();

await putUser(userTableEntry);

const originalArgument = faker.argumentTableEntry({
authorId: userTableEntry.userId,
});

await putArgument(originalArgument);

const event = {
body: {
argumentType: 'ActionArgument',
argument: faker.argumentForArgumentType(faker.attackingArgumentType()),
},
pathParameters: {
argumentId: originalArgument.argumentId,
},
headers: {
'mocked-user-id': userTableEntry.userId,
},
};
};

await expect(main(event)).rejects.toThrow(
'argument_does_not_match_argumentType'
);
});
});
});

```

## E.1.110 argupedia-api-master/src/functions/updateArgumentWithId/handler.ts

```
import { validateArgument } from '@helpers/argupedia';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';

import {
  getArgumentById,
  putArgument,
} from '../../../../../adapters/database/argumentTable';

import {
  Argument,
  ArgumentTableEntry,
  CreateNewArgumentHandlerBody,
} from '../../../../../types/arguments';

import schema from './schema';

const createNewAttackingArgument: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  const argumentId = event.pathParameters.argumentId;

  const originalArgument = await getArgumentById(argumentId);

  if (originalArgument === undefined) {
    throw new Error(`No argument with specified argumentId`);
  }
}
```

```

if (originalArgument.authorId !== userId) {
  throw new Error('User is not the author of the specified argument');
}

if (
  event.body.argument !== undefined &&
  event.body.argumentType !== undefined
) {
  validateArgument(event.body as CreateNewArgumentHandlerBody);
} else if (event.body.argument === undefined) {
  validateArgument({
    ...event.body,
    argumentType: originalArgument.argumentType,
  } as CreateNewArgumentHandlerBody);
} else if (event.body.argumentType === undefined) {
  validateArgument({
    ...event.body,
    argument: originalArgument.argument,
  } as CreateNewArgumentHandlerBody);
}

const modifiedDate = new Date().toISOString();

const argument = event.body.argument as Argument;

const argumentTableEntry: ArgumentTableEntry = {
  argumentId: event.pathParameters.argumentId,
  argumentType:
    event.body.argumentType === undefined
      ? originalArgument.argumentType
      : event.body.argumentType,
  argument:
    event.body.argument === undefined ? originalArgument.argument : argument,
  argumentText:
    event.body.argumentText === undefined
      ? originalArgument.argumentText
      : event.body.argumentText,
  argumentDepth: originalArgument.argumentDepth,
  authorId: userId,
  createdDate: originalArgument.createdDate,
  modifiedDate: modifiedDate,
}

```

```
    attacksArgumentId: originalArgument.attacksArgumentId,
    attackedByArgumentIds: originalArgument.attackedByArgumentIds,
    supportsArgumentId: originalArgument.supportsArgumentId,
    supportedByArgumentIds: originalArgument.supportedByArgumentIds,
    upvotedByUserIds: originalArgument.upvotedByUserIds,
    downvotedByUserIds: originalArgument.downvotedByUserIds,
    label: originalArgument.label,
    commentIds: originalArgument.commentIds,
};

await putArgument(argumentTableEntry);

return formatJSONResponse({
  message: argumentTableEntry,
  event,
});
};

export const main = middyfy(createNewAttackingArgument);
```

### E.1.111 argupedia-api-master/src/functions/updateArgumentWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

import schema from './schema';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'patch',
        path: 'rest/api/1/argument/{argumentId}',
        cors: true,
        request: {
          schemas: {
            'application/json': schema,
          },
        },
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.112 argupedia-api-master/src/functions/updateArgumentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

### E.1.113 argupedia-api-master/src/functions/updateArgumentWithId/schema.ts

```
export default {
  type: 'object',
  properties: {
    argumentType: { type: 'string' },
    argument: { type: 'object' },
    argumentText: { type: 'string' },
  },
} as const;
```

## E.1.114 argupedia-api-master/src/functions/updateCommentWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putComment } from '@adapters/database/commentTable';
import { putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { CommentTableEntry } from 'src/types/comments';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('updateCommentWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeAll(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
    main = (await import('./handler')).main;
  });

  afterAll(() => {
    jest.resetModules();
  });

  it('should_update_comment', async () => {
    const userTableEntry = faker.userTableEntry();

    await putUser(userTableEntry);

    const originalComment = faker.commentTableEntry({
      authorId: userTableEntry.userId,
    });
  });
});
```

```

await putComment(originalComment);

const event = {
  body: {
    commentBody: faker.random.words(30),
  },
  pathParameters: {
    commentId: originalComment.commentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as CommentTableEntry;

expect(entry.commentBody).toEqual(event.body.commentBody);
expect(entry.modifiedDate).not.toEqual(originalComment.modifiedDate);
});

it('should reject any operation where commentId is not valid', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const event = {
    pathParameters: {
      commentId: faker.datatype.uuid(),
    },
    headers: {
      'mocked-user-id': userTableEntry.userId,
    },
  };

  await expect(main(event)).rejects.toThrow(
    'No comment with specified commentId'
  );
});

```

```
it('should reject any operation where userId does not match authorId', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const originalComment = faker.commentTableEntry();

  await putComment(originalComment);

  const event = {
    pathParameters: {
      commentId: originalComment.commentId,
    },
    headers: {
      'mocked-user-id': userTableEntry.userId,
    },
  };

  await expect(main(event)).rejects.toThrow(
    'User is not the author of the specified comment'
  );
});

});
```

## E.1.115 argupedia-api-master/src/functions/updateCommentWithId/handler.ts

```
import { getCommentById, putComment } from '@adapters/database/commentTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';
import { CommentTableEntry } from 'src/types/comments';

import schema from './schema';

const updateCommentWithId: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    let payload: CognitoIdTokenPayload = undefined;

    if (
        event.headers !== undefined &&
        event.headers.Authorization !== undefined
    ) {
        payload = await verifyJwtToken(event.headers.Authorization.substring(7));
    }

    const isTest = process.env.JEST_WORKER_ID;
    const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

    const commentId = event.pathParameters.commentId;

    const originalComment = await getCommentById(commentId);

    if (originalComment === undefined) {
        throw new Error('No comment with specified commentId');
    }

    if (originalComment.authorId !== userId) {
        throw new Error('User is not the author of the specified comment');
    }

    const modifiedDate = new Date().toISOString();

    const commentTableEntry: CommentTableEntry = {
```

```
commentId: event.pathParameters.commentId,
argumentId: originalComment.argumentId,
authorId: userId,
commentBody: event.body.commentBody,
createdDate: originalComment.createdDate,
modifiedDate: modifiedDate,
};

await putComment(commentTableEntry);

return formatJSONResponse({
  message: commentTableEntry,
  event,
});
};

export const main = middyfy(updateCommentWithId);
```

### E.1.116 argupedia-api-master/src/functions/updateCommentWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

import schema from './schema';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'patch',
        path: 'rest/api/1/comment/{commentId}',
        cors: true,
        request: {
          schemas: {
            'application/json': schema,
          },
        },
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.117 argupedia-api-master/src/functions/updateCommentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.118 argupedia-api-master/src/functions/updateCommentWithId/schema.ts

```
export default {
  type: 'object',
  properties: {
    commentBody: { type: 'string' },
  },
  required: ['commentBody'],
} as const;
```

## E.1.119 argupedia-api-master/src/functions/updateUserWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import { putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { UserTableEntry } from 'src/types/users';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('updateUserWithId', () => {
  let main;
  let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

  beforeEach(async () => {
    mockedMiddyfy = mocked(middyfy);
    mockedMiddyfy.mockImplementation((handler: Handler) => {
      return handler as never;
    });
  });

  main = (await import('./handler')).main;
});

afterAll(() => {
  jest.resetModules();
});

it('should_update_user', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const originalUser = faker.userTableEntry({
    userId: userTableEntry.userId,
  });
});
```

```

await putUser(originalUser);

const event = {
  body: {
    biography: faker.random.words(30),
    displayName: faker.name.findName(),
    profilePictureURL: faker.image.imageUrl(),
  },
  pathParameters: {
    userId: userTableEntry.userId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

const { body } = await main(event);
const { message } = JSON.parse(body);
const entry = message as UserTableEntry;

expect(entry.biography).toEqual(event.body.biography);
expect(entry.displayName).toEqual(event.body.displayName);
expect(entry.profilePictureURL).toEqual(event.body.profilePictureURL);
};

it('should reject any operation where userId is not valid', async () => {
  const originalUser = faker.userTableEntry();

  await putUser(originalUser);

  const event = {
    body: {
      biography: faker.random.words(30),
      displayName: faker.name.findName(),
      profilePictureURL: faker.image.imageUrl(),
    },
    pathParameters: {
      userId: faker.datatype.uuid(),
    },
    headers: {
      'mocked-user-id': originalUser.userId,
    },
  };
}

```

```

};

await expect(main(event)).rejects.toThrow('No user with specified userId');
});

it('should reject any operation where userId does not match logged in user', async () => {
  const originalUser = faker.userTableEntry();

  await putUser(originalUser);

  const fakeUser = faker.userTableEntry();

  await putUser(fakeUser);

  const event = {
    body: {
      biography: faker.random.words(30),
      displayName: faker.name.findName(),
      profilePictureURL: faker.image.imageUrl(),
    },
    pathParameters: {
      userId: fakeUser.userId,
    },
    headers: {
      'mocked-user-id': originalUser.userId,
    },
  };

  await expect(main(event)).rejects.toThrow(
    'User cannot update a different user'
  );
});
});

```

## E.1.120 argupedia-api-master/src/functions/updateUserWithId/handler.ts

```
import { getUserId, putUser } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';
import { UserTableEntry } from 'src/types/users';

import schema from './schema';

const updateUserWithId: ValidatedEventAPIGatewayProxyEvent<
    typeof schema
> = async (event) => {
    let payload: CognitoIdTokenPayload = undefined;

    if (
        event.headers !== undefined &&
        event.headers.Authorization !== undefined
    ) {
        payload = await verifyJwtToken(event.headers.Authorization.substring(7));
    }

    const isTest = process.env.JEST_WORKER_ID;
    const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

    const originalUser = await getUserId(event.pathParameters.userId);

    if (originalUser === undefined) {
        throw new Error('No user with specified userId');
    }

    if (originalUser.userId !== userId) {
        throw new Error('User cannot update a different user');
    }

    const userTableEntry: UserTableEntry = {
        userId: event.pathParameters.userId,
        authoredArgumentIds: originalUser.authoredArgumentIds,
        authoredCommentIds: originalUser.authoredCommentIds,
        biography: event.body
            ? event.body.biography || originalUser.biography
            : originalUser.biography
    };

    await putUser(userTableEntry);
}
```

```
: originalUser.biography,  
displayName: event.body  
? event.body.displayName || originalUser.displayName  
: originalUser.displayName,  
lastSignedOn: originalUser.lastSignedOn,  
profilePictureURL: event.body  
? event.body.profilePictureURL || originalUser.profilePictureURL  
: originalUser.profilePictureURL,  
votedArgumentIds: originalUser.votedArgumentIds,  
};  
  
await putUser(userTableEntry);  
  
return formatJSONResponse({  
  message: userTableEntry,  
  event,  
});  
};  
  
export const main = middyfy(updateUserWithId);
```

### E.1.121 argupedia-api-master/src/functions/updateUserWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

import schema from './schema';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'patch',
        path: 'rest/api/1/user/{userId}',
        cors: true,
        request: {
          schemas: {
            'application/json': schema,
          },
        },
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.122 argupedia-api-master/src/functions/updateUserWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\":\"Frederic\"}"  
}
```

### E.1.123 argupedia-api-master/src/functions/updateUserWithId/schema.ts

```
export default {
  type: 'object',
  properties: {
    biography: { type: 'string' },
    displayName: { type: 'string' },
    profilePictureURL: { type: 'string' },
  },
} as const;
```

## E.1.124 argupedia-api-master/src/functions/upvoteArgumentWithId/handler.spec.ts

```
/* eslint-disable @typescript-eslint/no-unsafe-member-access */
/* eslint-disable @typescript-eslint/no-unsafe-argument */
/* eslint-disable @typescript-eslint/no-unsafe-call */
/* eslint-disable @typescript-eslint/no-unsafe-assignment */

import {
    addUserIdToArgumentDownvotes,
    getArgumentById,
    putArgument,
} from '@adapters/database/argumentTable';

import { getUserId, putUser } from '@adapters/database/userTable';
import { middyfy } from '@libs/lambda';
import { Handler } from 'aws-lambda';
import { mocked } from 'ts-jest/utils';

import faker from '../../helpers/faker';

jest.mock('@libs/lambda');

describe('upvoteArgumentWithId', () => {
    let main;
    let mockedMiddyfy: jest.MockedFunction<typeof middyfy>;

    beforeEach(async () => {
        mockedMiddyfy = mocked(middyfy);
        mockedMiddyfy.mockImplementation((handler: Handler) => {
            return handler as never;
        });
    });

    main = (await import('./handler')).main;
});

afterAll(() => {
    jest.resetModules();
});

it('should_add_user_id_to_argument_upvotes', async () => {
    const userTableEntry = faker.userTableEntry();

    await putUser(userTableEntry);
```

```

const argumentTableEntry = faker.argumentTableEntry();

await putArgument(argumentTableEntry);

expect(argumentTableEntry.upvotedByUserIds).not.toContain(
  userTableEntry.userId
);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await main(event);

const updatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(updatedArgumentTableEntry.upvotedByUserIds).toContain(
  userTableEntry.userId
);

const updatedUserTableEntry = await getUserById(userTableEntry.userId);

expect(updatedUserTableEntry.votedArgumentIds).toContain(
  argumentTableEntry.argumentId
);
});

it('should not add user id twice to upvotes', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const argumentTableEntry = faker.argumentTableEntry();

  await putArgument(argumentTableEntry);
}

```

```

expect(argumentTableEntry.upvotedByUserIds).not.toContain(
  userTableEntry.userId
);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await main(event);
await main(event);

const updatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

const upvoteSet = new Set(updatedArgumentTableEntry.upvotedByUserIds);

expect(updatedArgumentTableEntry.upvotedByUserIds.length).toEqual(
  upvoteSet.size
);
});

it('should_remove_user_id_from_downvotes_when_adding_to_upvotes', async () => {
  const userTableEntry = faker.userTableEntry();

  await putUser(userTableEntry);

  const argumentTableEntry = faker.argumentTableEntry();

  await putArgument(argumentTableEntry);

  expect(argumentTableEntry.upvotedByUserIds).not.toContain(
    userTableEntry.userId
);
  expect(argumentTableEntry.downvotedByUserIds).not.toContain(
    userTableEntry.userId
);
});

```

```

);

await addUserIdToArgumentDownvotes(
  argumentTableEntry.argumentId,
  userTableEntry.userId
);

const middleUpdatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(middleUpdatedArgumentTableEntry.upvotedByUserIds).not.toContain(
  userTableEntry.userId
);
expect(middleUpdatedArgumentTableEntry.downvotedByUserIds).toContain(
  userTableEntry.userId
);

const event = {
  pathParameters: {
    argumentId: argumentTableEntry.argumentId,
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await main(event);

const finalUpdatedArgumentTableEntry = await getArgumentById(
  argumentTableEntry.argumentId
);

expect(finalUpdatedArgumentTableEntry.upvotedByUserIds).toContain(
  userTableEntry.userId
);
expect(finalUpdatedArgumentTableEntry.downvotedByUserIds).not.toContain(
  userTableEntry.userId
);
});

it('should reject any operation where argumentId is not valid', async () => {

```

```
const userTableEntry = faker.userTableEntry();

await putUser(userTableEntry);

const event = {
  pathParameters: {
    argumentId: faker.datatype.uuid(),
  },
  headers: {
    'mocked-user-id': userTableEntry.userId,
  },
};

await expect(main(event)).rejects.toThrow(
  'No argument with specified argumentId'
);
});

});
```

## E.1.125 argupedia-api-master/src/functions/upvoteArgumentWithId/handler.ts

```
import { addArgumentIdForUserVotedArgumentIds } from '@adapters/database/userTable';
import { verifyJwtToken } from '@helpers/JWT';
import type { ValidatedEventAPIGatewayProxyEvent } from '@libs/apiGateway';
import { formatJSONResponse } from '@libs/apiGateway';
import { middyfy } from '@libs/lambda';
import { CognitoIdTokenPayload } from 'aws-jwt-verify/jwt-model';

import {
  addUserIdToArgumentUpvotes,
  getArgumentById,
  removeUserIdFromArgumentDownvotes,
} from '../../../../../adapters/database/argumentTable';
import schema from './schema';

const upvoteArgumentWithId: ValidatedEventAPIGatewayProxyEvent<
  typeof schema
> = async (event) => {
  let payload: CognitoIdTokenPayload = undefined;

  if (
    event.headers !== undefined &&
    event.headers.Authorization !== undefined
  ) {
    payload = await verifyJwtToken(event.headers.Authorization.substring(7));
  }

  const isTest = process.env.JEST_WORKER_ID;
  const userId = isTest ? event.headers['mocked-user-id'] : payload.sub;

  const argumentId = event.pathParameters.argumentId;

  const argumentTableEntry = await getArgumentById(argumentId);

  if (argumentTableEntry === undefined) {
    throw new Error(`No argument with specified argumentId`);
  }

  await removeUserIdFromArgumentDownvotes(argumentId, userId);
  await addUserIdToArgumentUpvotes(argumentId, userId);
  await addArgumentIdForUserVotedArgumentIds(userId, argumentId);
}
```

```
const argumentToBeReturned = await getArgumentById(argumentId);

return formatJSONResponse({
  message: argumentToBeReturned,
  event,
});

export const main = middyfy(upvoteArgumentWithId);
```

### E.1.126 argupedia-api-master/src/functions/upvoteArgumentWithId/index.ts

```
import { handlerPath } from '@libs/handlerResolver';

export default {
  handler: `${handlerPath(__dirname)}/handler.main`,
  events: [
    {
      http: {
        method: 'post',
        path: 'rest/api/1/argument/{argumentId}/upvote',
        cors: true,
        authorizer: {
          type: 'COGNITO_USER_POOLS',
          authorizerId: {
            Ref: 'ApiGatewayAuthorizer',
          },
        },
      },
    },
  ],
};
```

### E.1.127 argupedia-api-master/src/functions/upvoteArgumentWithId/mock.json

```
{  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"name\": \"Frederic\"}"  
}
```

### E.1.128 argupedia-api-master/src/functions/upvoteArgumentWithId/schema.ts

```
export default {} as const;
```

### E.1.129 argupedia-api-master/src/helpers/Cognito/index.ts

```
import {
  AdminDeleteUserCommand,
  CognitoIdentityProviderClient,
} from '@aws-sdk/client-cognito-identity-provider';

const client = new CognitoIdentityProviderClient({
  region: 'eu-west-2',
});

const { COGNITO_USER_POOL_ID } = process.env;

const deleteCognitoUserWithId = async (userId: string) => {
  return await client.send(
    new AdminDeleteUserCommand({
      UserPoolId: COGNITO_USER_POOL_ID,
      Username: userId,
    })
  );
};

export { deleteCognitoUserWithId };
```

### E.1.130 argupedia-api-master/src/helpers/DynamoDB/config.ts

```
const isTest = process.env.JEST_WORKER_ID;
const config = {
  convertEmptyValues: true,
  ...(isTest && {
    endpoint: 'localhost:8000',
    sslEnabled: false,
    region: 'local-env',
  }),
};

export { config };
```

### E.1.131 argupedia-api-master/src/helpers/DynamoDB/index.ts

```
import { config } from './config';

export default {
  config,
};
```

### E.1.132 argupedia-api-master/src/helpers/JWT/index.test.ts

```
// import { verifyJwtToken } from './';

// it('should decode jwt token', async () => {
//   const token =
//     'eyJraWQiOiJrYitvQOMrUlNXOVowMlZ5TldWS0tFMUtsSThKQThXQStmRnIy0VRpeTEOPSIiImFsZyI6IlJTMjU2In0.
eyJzdWIiOiI1YTQyMzJjMC00OWY1LTrjMTctOTZhYS1jODk5MDZiNmQ1MDQiLCJlbWFpbF92ZXJpZmlIZCI6dHJ1ZSwiaXNzIjoiaHR0cHM6XC9cL2NvZ25pdG8taWRwLmV1L
.Juevhrl2z1qE6Dt6gb3tF_j1MwIClXZKqLH4qfKk6Z8dBpWNy1Ro4YiwU5UN-
KlGnB_s4un2vhSfIQuAvDDYWGKuCTUiFocPBxTuzenbeRGqKUy-5IT3Fgiu_Kr5SNN7-
r1yTF0uqNOVTHrc1HEy7V5W30CWBn91QyyekpsXfYrAvu1U18RseiOBQAgsS33cU6yA_RBGeMKcza90nuJ1A50Qq-
q3H8Wan5SlZX4VRsZApoJUYXrieNGCryFl_CyshuKOZE0tGhjxYq4fkN9LgTA1y-ehN3EyZDQcHP5oZNCsoXbcxCuLuq20yAi8oPnKw3Q8rlb
-aTjPa7e6_ynexGA';

//   await expect(verifyJwtToken(token)).resolves.not.toThrow();
// });

it('should_pass', () => {
  expect(1).toBe(1);
});
```

### E.1.133 argupedia-api-master/src/helpers/JWT/index.ts

```
import { CognitoJwtVerifier } from 'aws-jwt-verify';

const { COGNITO_USER_POOL_ID, COGNITO_APP_CLIENT_ID } = process.env;

const verifyJwtToken = async (token: string) => {
  try {
    const verifier = CognitoJwtVerifier.create({
      userPoolId: COGNITO_USER_POOL_ID,
      tokenUse: 'id',
      clientId: COGNITO_APP_CLIENT_ID,
    });
    const payload = await verifier.verify(token);

    return payload;
  } catch {
    throw new Error('Token not valid');
  }
};

export { verifyJwtToken };
```

### E.1.134 argupedia-api-master/src/helpers/argupedia/index.ts

```
import labelArguments from './labelArguments';
import validateArgument from './validateArgument';

export { labelArguments, validateArgument };
```

## E.1.135 argupedia-api-master/src/helpers/argupedia/labelArguments.test.ts

```
import {
  getArgumentById,
  putArgument,
} from '../adapters/database/argumentTable';
import faker from '../faker';
import labelArguments from './labelArguments';

it('should_label_argument_one_attacker_as_OUT', async () => {
  const argument = faker.argumentTableEntry({
    attacksArgumentId: undefined,
    argumentDepth: 0,
    supportedByArgumentIds: [],
  });

  const attackingArgument = faker.argumentTableEntry({
    attacksArgumentId: argument.argumentId,
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
    argumentDepth: 1,
    label: 'IN',
  });

  argument.attackedByArgumentIds = [attackingArgument.argumentId];

  await putArgument(argument);
  await putArgument(attackingArgument);

  await labelArguments(attackingArgument.argumentId);

  const finalArgument = await getArgumentById(argument.argumentId);

  expect(finalArgument.label).toEqual('OUT');
});

it('should_label_argument_with_multiple_attackers, where one is IN and others are OUT as OUT', async () => {
  const argument = faker.argumentTableEntry({
    attacksArgumentId: undefined,
    argumentDepth: 0,
    supportedByArgumentIds: [],
  });
}
```

```

const attackingArgument1 = faker.argumentTableEntry({
  attacksArgumentId: argument.argumentId,
  attackedByArgumentIds: [],
  supportedByArgumentIds: [],
  argumentDepth: 1,
  label: 'IN',
});

const attackingArgument2 = faker.argumentTableEntry({
  attacksArgumentId: argument.argumentId,
  attackedByArgumentIds: [],
  supportedByArgumentIds: [],
  argumentDepth: 1,
  label: 'IN',
});

const attackingAttackingArgument2 = faker.argumentTableEntry({
  attacksArgumentId: attackingArgument2.argumentId,
  attackedByArgumentIds: [],
  supportedByArgumentIds: [],
  argumentDepth: 2,
  label: 'IN',
});

argument.attackedByArgumentIds = [
  attackingArgument1.argumentId,
  attackingArgument2.argumentId,
];

attackingArgument2.attackedByArgumentIds = [
  attackingAttackingArgument2.argumentId,
];

await putArgument(argument);
await putArgument(attackingArgument1);
await putArgument(attackingArgument2);
await putArgument(attackingAttackingArgument2);

await labelArguments(attackingAttackingArgument2.argumentId);

const finalArgument = await getArgumentById(argument.argumentId);

```

```

expect(finalArgument.label).toEqual('OUT');

});

it('should_label_argument_with_multiple_attackers, where all are OUT as IN', async () => {
  const argument = faker.argumentTableEntry({
    attacksArgumentId: undefined,
    supportedByArgumentIds: [],
    argumentDepth: 0,
    label: 'OUT',
  });

  const attackingArgument1 = faker.argumentTableEntry({
    attacksArgumentId: argument.argumentId,
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
    argumentDepth: 1,
    label: 'OUT',
  });

  const attackingArgument2 = faker.argumentTableEntry({
    attacksArgumentId: argument.argumentId,
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
    argumentDepth: 1,
    label: 'IN',
  });

  const attackingAttackingArgument1 = faker.argumentTableEntry({
    attacksArgumentId: attackingArgument1.argumentId,
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
    argumentDepth: 2,
    label: 'IN',
  });

  const attackingAttackingArgument2 = faker.argumentTableEntry({
    attacksArgumentId: attackingArgument2.argumentId,
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
    argumentDepth: 2,
    label: 'IN',
  });
}

```

```

});;

argument.attackedByArgumentIds = [
  attackingArgument1.argumentId,
  attackingArgument2.argumentId,
];

attackingArgument1.attackedByArgumentIds = [
  attackingAttackingArgument1.argumentId,
];

attackingArgument2.attackedByArgumentIds = [
  attackingAttackingArgument2.argumentId,
];

await putArgument(argument);
await putArgument(attackingArgument1);
await putArgument(attackingArgument2);
await putArgument(attackingAttackingArgument1);
await putArgument(attackingAttackingArgument2);

await labelArguments(attackingAttackingArgument2.argumentId);

const finalArgument = await getArgumentById(argument.argumentId);

expect(finalArgument.label).toEqual('IN');

});

it('should_label_argument_with_no_attackers_as_IN', async () => {
  const argument = faker.argumentTableEntry({
    attacksArgumentId: undefined,
    argumentDepth: 0,
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
  });

  await putArgument(argument);

  await labelArguments(argument.argumentId);

  const finalArgument = await getArgumentById(argument.argumentId);
}

```

```

expect(finalArgument.label).toEqual('IN');

});

it('should_label_an_argument_with_the_same_label_as_the_argument_it_supports', async () => {
  const argument = faker.argumentTableEntry({
    attacksArgumentId: undefined,
    argumentDepth: 0,
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
  });

  const supportedArgument = faker.argumentTableEntry({
    attacksArgumentId: undefined,
    argumentDepth: 0,
    attackedByArgumentIds: [],
    supportedByArgumentIds: [],
    label: 'OUT',
    supportsArgumentId: undefined,
  });

  argument.supportsArgumentId = supportedArgument.argumentId;
  supportedArgument.supportedByArgumentIds = [argument.argumentId];

  await putArgument(argument);
  await putArgument(supportedArgument);

  await labelArguments(argument.argumentId);

  const finalArgument = await getArgumentById(argument.argumentId);

  expect(finalArgument.label).toEqual(supportedArgument.label);
});

```

## E.1.136 argupedia-api-master/src/helpers/argupedia/labelArguments.ts

```
import { ArgumentLabel } from 'src/types/arguments';

import {
  getArgumentById,
  putArgument,
} from '../../adapters/database/argumentTable';

/*
 * 
 * x is legally IN iff x is labelled IN and every y that attacks x is labelled OUT
 * x is legally OUT iff x is labelled OUT and there is at least one y that attacks x
 * and y is labelled IN
 * x is legally UNDEC iff x is labelled UNDEC and not every y that attacks x is labelled
 * OUT, and there is no y that attacks x such that y is labelled IN
 */
const labelArgumentAndSupportingArguments = async (
  argumentId: string,
  label: ArgumentLabel
) => {
  // Retrieve argument from database
  const argument = await getArgumentById(argumentId);

  // Set label of argument to provided label
  argument.label = label;

  // Store the argument back in the database
  await putArgument(argument);

  // For each child argument, label it the same as the parent
  for await (const supportedByArgumentId of argument.supportedByArgumentIds) {
    await labelArgumentAndSupportingArguments(supportedByArgumentId, label);
  }
};

const labelArguments = async (leafArgumentId: string) => {
  // Retrieve argument from database
  const argument = await getArgumentById(leafArgumentId);

  // Assume that the argument is in by default
  let argumentLabel: ArgumentLabel = 'IN';
```

```

// If the argument supports an argument, label it the same as the argument it supports
if (argument.supportsArgumentId !== undefined) {
  const supportingArgument = await getArgumentById(
    argument.supportsArgumentId
  );

  argumentLabel = supportingArgument.label;
}

// Go through all attackers of the argument
for await (const attackedByArgumentId of argument.attackedByArgumentIds) {
  const attackingArgument = await getArgumentById(attackedByArgumentId);

  // If an attacking argument is UNDECIDED, label the argument as UNDECIDED
  if (attackingArgument.label === 'UNDECIDED') {
    argumentLabel = 'UNDECIDED';
  }

  // If any attacking argument is IN, label the argument OUT
  if (attackingArgument.label === 'IN') {
    argumentLabel = 'OUT';
    break;
  }
}

// Label any arguments which support the argument
await labelArgumentAndSupportingArguments(leafArgumentId, argumentLabel);

// Continue up the tree and label the argument's parent argument
if (argument.attacksArgumentId !== undefined) {
  await labelArguments(argument.attacksArgumentId);
}
};

export default labelArguments;

```

### E.1.137 argupedia-api-master/src/helpers/argupedia/validateArgument.ts

```
import { CreateNewArgumentHandlerBody } from 'src/types/arguments';

const argumentDoesNotMatchArgumentType = new Error(
  'argument does not match argument type'
);

const validateArgument = (body: CreateNewArgumentHandlerBody) => {
  switch (body.argumentType) {
    case 'ActionArgument': {
      if (
        !(‘circumstance’ in body.argument) ||
        !(‘action’ in body.argument) ||
        !(‘newCircumstance’ in body.argument) ||
        !(‘goal’ in body.argument) ||
        !(‘value’ in body.argument)
      )
        throw argumentDoesNotMatchArgumentType;
      break;
    }
    case 'ArgumentFromPositionToKnow': {
      if (
        !(‘person’ in body.argument) ||
        !(‘fact’ in body.argument) ||
        !(‘proposition’ in body.argument)
      )
        throw argumentDoesNotMatchArgumentType;
      break;
    }
    case 'AppealToExpertOpinion': {
      if (
        !(‘expert’ in body.argument) ||
        !(‘domain’ in body.argument) ||
        !(‘fact’ in body.argument) ||
        !(‘proposition’ in body.argument)
      )
        throw argumentDoesNotMatchArgumentType;
      break;
    }
    case 'AppealToPopularOpinion': {
      if (!(‘fact’ in body.argument) || !(‘proposition’ in body.argument))
        throw argumentDoesNotMatchArgumentType;
    }
  }
}
```

```

        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'ArgumentFromAnalogy': {
    if (
        !('originalCase' in body.argument) ||
        !('similarCase' in body.argument) ||
        !('fact' in body.argument) ||
        !('proposition' in body.argument)
    )
        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'ArgumentFromCorrelationToCause': {
    if (!('cause' in body.argument) || !('effect' in body.argument))
        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'ArgumentFromPositiveOrNegativeConsequences': {
    if (
        !('cause' in body.argument) ||
        !('effect' in body.argument) ||
        !('consequence' in body.argument)
    )
        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'SlipperySlopeArgument': {
    if (
        !('firstStepPremise' in body.argument) ||
        !('recursivePremise' in body.argument) ||
        !('consequence' in body.argument)
    )
        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'ArgumentFromSign': {
    if (
        !('finding' in body.argument) ||
        !('fact' in body.argument) ||
        !('proposition' in body.argument)
    )

```

```

        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'ArgumentFromCommitment': {
    if (
        !('entity' in body.argument) ||
        !('existingCommitment' in body.argument) ||
        !('newCommitment' in body.argument)
    )
        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'ArgumentFromInconsistentCommitment': {
    if (
        !('entity' in body.argument) ||
        !('commitment' in body.argument) ||
        !('otherEvidence' in body.argument)
    )
        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'DirectAdHominemArgument': {
    if (!('entity' in body.argument) || !('criticism' in body.argument))
        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'CircumstantialAdHominemArgument': {
    if (
        !('entity' in body.argument) ||
        !('originalArgument' in body.argument) ||
        !('otherEvidence' in body.argument)
    )
        throw argumentDoesNotMatchArgumentType;
    break;
}

case 'ArgumentFromVerbalClassification': {
    if (
        !('entity' in body.argument) ||
        !('property' in body.argument) ||
        !('classification' in body.argument)
    )
        throw argumentDoesNotMatchArgumentType;
}

```

```
        break;
    }
    default:
        throw new Error('Invalid argument type');
    }
};

export default validateArgument;
```

### E.1.138 argupedia-api-master/src/helpers/faker/argument.ts

```
import faker from '@faker-js/faker';
import merge from 'deepmerge';
import { RecursivePartial } from 'src/types/recursivePartial';

import { mergeOptions } from '../../../../../helpers/mergeOptions';
import {

  ActionArgument,
  AppealToExpertOpinion,
  AppealToPopularOpinion,
  Argument,
  ArgumentFromAnalogy,
  ArgumentFromCommitment,
  ArgumentFromCorrelationToCause,
  ArgumentFromInconsistentCommitment,
  ArgumentFromPositionToKnow,
  ArgumentFromPositiveOrNegativeConsequences,
  ArgumentFromSign,
  ArgumentFromVerbalClassification,
  ArgumentTableEntry,
  argumentTypes,
  CircumstantialAdHominemArgument,
  DirectAdHominemArgument,
  SlipperySlopeArgument,
} from '../../../../../types/arguments';

const actionArgument = (
  actionArgument?: RecursivePartial<ActionArgument>
): ActionArgument =>
  merge(
    {
      circumstance: faker.random.words(3),
      action: faker.random.words(3),
      newCircumstance: faker.random.words(3),
      goal: faker.random.words(3),
      value: faker.random.words(3),
    },
    { ...actionArgument as ActionArgument },
    mergeOptions
  );

```

```

const argumentFromPositionToKnow = (
  argumentFromPositionToKnow?: RecursivePartial<ArgumentFromPositionToKnow>
): ArgumentFromPositionToKnow =>
  merge(
    {
      person: faker.random.words(3),
      fact: faker.random.words(3),
      proposition: faker.datatype.boolean(),
    },
    { ... (argumentFromPositionToKnow as ArgumentFromPositionToKnow) },
    mergeOptions
  );
}

const appealToExpertOpinion = (
  appealToExpertOpinion?: RecursivePartial<AppealToExpertOpinion>
): AppealToExpertOpinion =>
  merge(
    {
      expert: faker.random.words(3),
      domain: faker.random.words(3),
      fact: faker.random.words(3),
      proposition: faker.datatype.boolean(),
    },
    { ... (appealToExpertOpinion as AppealToExpertOpinion) },
    mergeOptions
  );
}

const appealToPopularOpinion = (
  appealToPopularOpinion?: RecursivePartial<AppealToPopularOpinion>
): AppealToPopularOpinion =>
  merge(
    {
      fact: faker.random.words(3),
      proposition: faker.datatype.boolean(),
    },
    { ... (appealToPopularOpinion as AppealToPopularOpinion) },
    mergeOptions
  );
}

const argumentFromAnalogy = (
  argumentFromAnalogy?: RecursivePartial<ArgumentFromAnalogy>
): ArgumentFromAnalogy =>

```

```

merge(
{
  originalCase: faker.random.words(3),
  similarCase: faker.random.words(3),
  fact: faker.random.words(3),
  proposition: faker.datatype.boolean(),
},
{ ... (argumentFromAnalogy as ArgumentFromAnalogy) },
mergeOptions
);

const argumentFromCorrelationToCause = (
  argumentFromCorrelationToCause?: RecursivePartial<ArgumentFromCorrelationToCause>
): ArgumentFromCorrelationToCause =>
  merge(
{
  cause: faker.random.words(3),
  effect: faker.random.words(3),
},
{ ... (argumentFromCorrelationToCause as ArgumentFromCorrelationToCause) },
mergeOptions
);

const argumentFromPositiveOrNegativeConsequences = (
  argumentFromPositiveOrNegativeConsequences?: RecursivePartial<ArgumentFromPositiveOrNegativeConsequences>
): ArgumentFromPositiveOrNegativeConsequences =>
  merge(
{
  cause: faker.random.words(3),
  effect: faker.random.words(3),
  consequence: ['GOOD', 'BAD'][faker.datatype.number(1)] as 'GOOD' | 'BAD',
},
{
  ... (argumentFromPositiveOrNegativeConsequences as ArgumentFromPositiveOrNegativeConsequences),
},
mergeOptions
);

const slipperySlopeArgument = (
  slipperySlopeArgument?: RecursivePartial<SlipperySlopeArgument>
): SlipperySlopeArgument =>
  merge(

```

```

{
  firstStepPremise: faker.random.words(3),
  recursivePremise: faker.random.words(3),
  consequence: ['GOOD', 'BAD'][faker.datatype.number(1)] as 'GOOD' | 'BAD',
},
{ ... (slipperySlopeArgument as SlipperySlopeArgument) },
mergeOptions
);

const argumentFromSign = (
  argumentFromSign?: RecursivePartial<ArgumentFromSign>
): ArgumentFromSign =>
  merge(
  {
    finding: faker.random.words(3),
    fact: faker.random.words(3),
    proposition: faker.datatype.boolean(),
  },
  { ... (argumentFromSign as ArgumentFromSign) },
  mergeOptions
);

const argumentFromCommitment = (
  argumentFromCommitment?: RecursivePartial<ArgumentFromCommitment>
): ArgumentFromCommitment =>
  merge(
  {
    entity: faker.random.words(3),
    existingCommitment: faker.random.words(3),
    newCommitment: faker.random.words(3),
  },
  { ... (argumentFromCommitment as ArgumentFromCommitment) },
  mergeOptions
);

const argumentFromInconsistentCommitment = (
  argumentFromInconsistentCommitment?: RecursivePartial<ArgumentFromInconsistentCommitment>
): ArgumentFromInconsistentCommitment =>
  merge(
  {
    entity: faker.random.words(3),
    commitment: faker.random.words(3),
  }
);

```

```

otherEvidence: faker.random.words(3),
},
{
  ... (argumentFromInconsistentCommitment as ArgumentFromInconsistentCommitment),
},
mergeOptions
);

const directAdHominemArgument = (
  directAdHominemArgument?: RecursivePartial<DirectAdHominemArgument>
): DirectAdHominemArgument =>
  merge(
  {
    entity: faker.random.words(3),
    criticism: faker.random.words(3),
  },
  { ... (directAdHominemArgument as DirectAdHominemArgument) },
  mergeOptions
);

const circumstantialAdHominemArgument = (
  circumstantialAdHominemArgument?: RecursivePartial<CircumstantialAdHominemArgument>
): CircumstantialAdHominemArgument =>
  merge(
  {
    entity: faker.random.words(3),
    originalArgument: faker.random.words(3),
    otherEvidence: faker.random.words(3),
  },
  { ... (circumstantialAdHominemArgument as CircumstantialAdHominemArgument) },
  mergeOptions
);

const argumentFromVerbalClassification = (
  argumentFromVerbalClassification?: RecursivePartial<ArgumentFromVerbalClassification>
): ArgumentFromVerbalClassification =>
  merge(
  {
    entity: faker.random.words(3),
    property: faker.random.words(3),
    classification: faker.random.words(3),
  },

```

```

{
  ... (argumentFromVerbalClassification as ArgumentFromVerbalClassification),
},
mergeOptions
);

const argumentForArgumentType = (argumentType: string): Argument => {
  switch (argumentType) {
    case 'ActionArgument':
      return actionArgument();
    case 'ArgumentFromPositionToKnow':
      return argumentFromPositionToKnow();
    case 'AppealToExpertOpinion':
      return appealToExpertOpinion();
    case 'AppealToPopularOpinion':
      return appealToPopularOpinion();
    case 'ArgumentFromAnalogy':
      return argumentFromAnalogy();
    case 'ArgumentFromCorrelationToCause':
      return argumentFromCorrelationToCause();
    case 'ArgumentFromPositiveOrNegativeConsequences':
      return argumentFromPositiveOrNegativeConsequences();
    case 'SlipperySlopeArgument':
      return slipperySlopeArgument();
    case 'ArgumentFromSign':
      return argumentFromSign();
    case 'ArgumentFromCommitment':
      return argumentFromCommitment();
    case 'ArgumentFromInconsistentCommitment':
      return argumentFromInconsistentCommitment();
    case 'DirectAdHominemArgument':
      return directAdHominemArgument();
    case 'CircumstantialAdHominemArgument':
      return circumstantialAdHominemArgument();
    case 'ArgumentFromVerbalClassification':
      return argumentFromVerbalClassification();
    default:
      throw new Error(`Unknown argument type: ${argumentType}`);
  }
};

const argumentType = (): string => {

```

```

return argumentTypes[Math.floor(Math.random() * argumentTypes.length)];
};

const attackingArgumentType = (): string => {
  const argumentTypes = [
    'ArgumentFromPositionToKnow',
    'AppealToExpertOpinion',
    'AppealToPopularOpinion',
    'ArgumentFromAnalogy',
    'ArgumentFromCorrelationToCause',
    'ArgumentFromPositiveOrNegativeConsequences',
    'SlipperySlopeArgument',
  ];
}

return argumentTypes[Math.floor(Math.random() * argumentTypes.length)];
};

const argument = (): Argument => {
  const possibleArguments = [
    actionArgument,
    argumentFromPositionToKnow,
    appealToExpertOpinion,
    appealToPopularOpinion,
    argumentFromAnalogy,
    argumentFromCorrelationToCause,
    argumentFromPositiveOrNegativeConsequences,
    slipperySlopeArgument,
    argumentFromSign,
    argumentFromCommitment,
    argumentFromInconsistentCommitment,
    directAdHominemArgument,
    circumstantialAdHominemArgument,
    argumentFromVerbalClassification,
  ];
}

return possibleArguments[
  Math.floor(Math.random() * possibleArguments.length)
]();
};

const argumentTableEntry = (
  argumentTableEntry?: RecursivePartial<ArgumentTableEntry>

```

```

): ArgumentTableEntry => {
  const selectedArgumentType = argumentType();

  return merge(
  {
    argumentId: faker.datatype.uuid(),
    argumentType: selectedArgumentType,
    argument: argumentTableEntry
    ? {}
    : argumentForArgumentType(selectedArgumentType),
    authorId: faker.datatype.uuid(),
    createdDate: faker.date.past().toISOString(),
    modifiedDate: faker.date.past().toISOString(),
    attacksArgumentId: undefined,
    supportsArgumentId: undefined,
    supportedByArgumentIds: Array.from(
      { length: faker.datatype.number(20) },
      () => faker.datatype.uuid()
    ),
    attackedByArgumentIds: Array.from(
      { length: faker.datatype.number(20) },
      () => faker.datatype.uuid()
    ),
    upvotedByUserIds: Array.from({ length: faker.datatype.number(20) }, () =>
      faker.datatype.uuid()
    ),
    downvotedByUserIds: Array.from(
      { length: faker.datatype.number(20) },
      () => faker.datatype.uuid()
    ),
    label: 'UNDECIDED',
    commentIds: Array.from({ length: faker.datatype.number(20) }, () =>
      faker.datatype.uuid()
    ),
    argumentDepth: 0,
  },
  { ... (argumentTableEntry as ArgumentTableEntry) },
  mergeOptions
);
};

export {

```

```
actionArgument,  
appealToExpertOpinion,  
appealToPopularOpinion,  
argument,  
argumentForArgumentType,  
argumentFromAnalogy,  
argumentFromCommitment,  
argumentFromCorrelationToCause,  
argumentFromInconsistentCommitment,  
argumentFromPositionToKnow,  
argumentFromPositiveOrNegativeConsequences,  
argumentFromSign,  
argumentFromVerbalClassification,  
argumentTableEntry,  
argumentType,  
attackingArgumentType,  
circumstantialAdHominemArgument,  
directAdHominemArgument,  
slipperySlopeArgument,  
};
```

### E.1.139 argupedia-api-master/src/helpers/faker/comment.ts

```
import faker from '@faker-js/faker';
import merge from 'deepmerge';
import { CommentTableEntry } from 'src/types/comments';
import { RecursivePartial } from 'src/types/recursivePartial';

import { mergeOptions } from '../../helpers/mergeOptions';

const commentTableEntry = (
  commentTableEntry?: RecursivePartial<CommentTableEntry>
): CommentTableEntry =>
  merge(
    {
      commentId: faker.datatype.uuid(),
      commentBody: faker.random.words(10),
      createdDate: faker.date.past().toISOString(),
      modifiedDate: faker.date.past().toISOString(),
      authorId: faker.datatype.uuid(),
      argumentId: faker.datatype.uuid(),
    },
    { ...commentTableEntry as CommentTableEntry },
    mergeOptions
  );

export { commentTableEntry };
```

## E.1.140 argupedia-api-master/src/helpers/faker/index.ts

```
import faker from '@faker-js/faker';

import {
  actionArgument,
  appealToExpertOpinion,
  appealToPopularOpinion,
  argument,
  argumentForArgumentType,
  argumentFromAnalogy,
  argumentFromCorrelationToCause,
  argumentFromPositionToKnow,
  argumentFromPositiveOrNegativeConsequences,
  argumentTableEntry,
  argumentType,
  attackingArgumentType,
  slipperySlopeArgument,
} from './argument';

import { commentTableEntry } from './comment';
import { userTableEntry } from './user';

export default {
  ...faker,
  actionArgument,
  appealToExpertOpinion,
  appealToPopularOpinion,
  argument,
  argumentFromAnalogy,
  argumentFromCorrelationToCause,
  argumentFromPositionToKnow,
  argumentFromPositiveOrNegativeConsequences,
  argumentTableEntry,
  slipperySlopeArgument,
  commentTableEntry,
  userTableEntry,
  attackingArgumentType,
  argumentType,
  argumentForArgumentType,
};

};
```

### E.1.141 argupedia-api-master/src/helpers/faker/user.ts

```
import faker from '@faker-js/faker';
import merge from 'deepmerge';
import { RecursivePartial } from 'src/types/recursivePartial';
import { UserTableEntry } from 'src/types/users';

import { mergeOptions } from '../../helpers/mergeOptions';

const userTableEntry = (
  userTableEntry?: RecursivePartial<UserTableEntry>
): UserTableEntry =>
  merge(
    {
      userId: faker.datatype.uuid(),
      authoredArgumentIds: Array.from(
        { length: faker.datatype.number(20) },
        () => faker.datatype.uuid()
      ),
      authoredCommentIds: Array.from(
        { length: faker.datatype.number(20) },
        () => faker.datatype.uuid()
      ),
      votedArgumentIds: Array.from({ length: faker.datatype.number(20) }, () =>
        faker.datatype.uuid()
      ),
      displayName: faker.name.findName(),
      biography: faker.lorem.paragraph(),
      profilePictureURL: faker.image.avatar(),
      lastSignedOn: faker.date.past().toISOString(),
    },
    {
      ... (userTableEntry as UserTableEntry),
      mergeOptions
    };
  );

export { userTableEntry };
```

### E.1.142 argupedia-api-master/src/helpers/mergeOptions/index.ts

```
export * from './mergeOptions';
```

### E.1.143 argupedia-api-master/src/helpers/mergeOptions/mergeOptions.ts

```
import merge from 'deepmerge';

const mergeOptions: merge.Options = {
  // eslint-disable-next-line @typescript-eslint/no-unsafe-return
  arrayMerge: (_destinationArray, sourceArray, _options) => sourceArray,
};

export { mergeOptions };
```

## E.1.144 argupedia-api-master/src/libs/apiGateway.ts

```
import type {
  APIGatewayProxyEvent,
  APIGatewayProxyResult,
  Handler,
} from 'aws-lambda';

import type { FromSchema } from 'json-schema-to-ts';

type ValidatedAPIGatewayProxyEvent<S> = Omit<APIGatewayProxyEvent, 'body'> & {
  body: FromSchema<S>;
};

export type ValidatedEventAPIGatewayProxyEvent<S> = Handler<
  ValidatedAPIGatewayProxyEvent<S>,
  APIGatewayProxyResult
>;

export const formatJSONResponse = (response: Record<string, unknown>) => {
  // let origin = 'https://d3326mk0xvlcgk.amplifyapp.com/';

  // if (response.event !== undefined) {
  //   type Event = { headers: { origin: string } };
  //   const event = response.event as Event;

  //   if (
  //     event.headers !== undefined &&
  //     event.headers.origin === 'http://localhost:8100'
  //   ) {
  //     origin = 'http://localhost:8100';
  //   }
  // }

  // //access-control-allow-origin is set to * so that this may be compiled and tested by others
  return {
    statusCode: 200,
    headers: {
      // 'Access-Control-Allow-Origin': origin,
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Credentials': true,
    },
    body: JSON.stringify(response),
  };
}
```

};

### E.1.145 argupedia-api-master/src/libs/handlerResolver.ts

```
export const handlerPath = (context: string) => {
  return `${context.split(process.cwd())[1].substring(1).replace(/\\"g, '/')}`;
};
```

### E.1.146 argupedia-api-master/src/libs/lambda.ts

```
import middy from '@middy/core';
import middyJsonBodyParser from '@middy/http-json-body-parser';

export const middyfy = (handler) => {
  // eslint-disable-next-line @typescript-eslint/no-unsafe-argument
  return middy(handler).use(middyJsonBodyParser());
};
```

## E.1.147 argupedia-api-master/src/types/arguments.ts

```
type ActionArgument = {
    circumstance: string;
    action: string;
    newCircumstance: string;
    goal: string;
    value: string;
};

type ArgumentFromPositionToKnow = {
    person: string;
    fact: string;
    proposition: boolean;
};

type AppealToExpertOpinion = {
    expert: string;
    domain: string;
    fact: string;
    proposition: boolean;
};

type AppealToPopularOpinion = {
    fact: string;
    proposition: boolean;
};

type ArgumentFromAnalogy = {
    originalCase: string;
    similarCase: string;
    fact: string;
    proposition: boolean;
};

type ArgumentFromCorrelationToCause = {
    cause: string;
    effect: string;
};

type ArgumentFromPositiveOrNegativeConsequences = {
    cause: string;
}
```

```

effect: string;
consequence: 'GOOD' | 'BAD';
};

type SlipperySlopeArgument = {
  firstStepPremise: string;
  recursivePremise: string;
  consequence: 'GOOD' | 'BAD';
};

type ArgumentFromSign = {
  finding: string;
  fact: string;
  proposition: boolean;
};

type ArgumentFromCommitment = {
  entity: string;
  existingCommitment: string;
  newCommitment: string;
};

type ArgumentFromInconsistentCommitment = {
  entity: string;
  commitment: string;
  otherEvidence: string;
};

type DirectAdHominemArgument = {
  entity: string;
  criticism: string;
};

type CircumstantialAdHominemArgument = {
  entity: string;
  originalArgument: string;
  otherEvidence: string;
};

type ArgumentFromVerbalClassification = {
  entity: string;
  property: string;
};

```

```

classification: string;
};

type Argument =
| ActionArgument
| ArgumentFromPositionToKnow
| AppealToExpertOpinion
| AppealToPopularOpinion
| ArgumentFromAnalogy
| ArgumentFromCorrelationToCause
| ArgumentFromPositiveOrNegativeConsequences
| SlipperySlopeArgument
| ArgumentFromSign
| ArgumentFromCommitment
| ArgumentFromInconsistentCommitment
| DirectAdHominemArgument
| CircumstantialAdHominemArgument
| ArgumentFromVerbalClassification;

const argumentTypes = [
  'ActionArgument',
  'ArgumentFromPositionToKnow',
  'AppealToExpertOpinion',
  'AppealToPopularOpinion',
  'ArgumentFromAnalogy',
  'ArgumentFromCorrelationToCause',
  'ArgumentFromPositiveOrNegativeConsequences',
  'SlipperySlopeArgument',
  'ArgumentFromSign',
  'ArgumentFromCommitment',
  'ArgumentFromInconsistentCommitment',
  'DirectAdHominemArgument',
  'CircumstantialAdHominemArgument',
  'ArgumentFromVerbalClassification',
];

type ArgumentLabel = 'IN' | 'OUT' | 'UNDECIDED';

type ArgumentTableEntry = {
  argumentId: string;
  argumentType: string;
  argument: Argument;
}

```

```

argumentText?: string;
authorId: string;
createdDate: string;
modifiedDate: string;
supportsArgumentId?: string;
supportedByArgumentIds: Array<string>;
attacksArgumentId?: string;
attackedByArgumentIds: Array<string>;
upvotedByUserIds: Array<string>;
downvotedByUserIds: Array<string>;
label: ArgumentLabel;
commentIds: Array<string>;
argumentDepth: number;
};


```

```

type CreateNewArgumentHandlerBody = {
  argumentType: string;
  argument: Argument;
};

export {
  ActionArgument,
  AppealToExpertOpinion,
  AppealToPopularOpinion,
  Argument,
  ArgumentFromAnalogy,
  ArgumentFromCommitment,
  ArgumentFromCorrelationToCause,
  ArgumentFromInconsistentCommitment,
  ArgumentFromPositionToKnow,
  ArgumentFromPositiveOrNegativeConsequences,
  ArgumentFromSign,
  ArgumentFromVerbalClassification,
  ArgumentLabel,
  ArgumentTableEntry,
  argumentTypes,
  CircumstantialAdHominemArgument,
  CreateNewArgumentHandlerBody,
  DirectAdHominemArgument,
  SlipperySlopeArgument,
};


```

### E.1.148 argupedia-api-master/src/types/comments.ts

```
type CommentTableEntry = {  
    commentId: string;  
    commentBody: string;  
    createdDate: string;  
    modifiedDate: string;  
    authorId: string;  
    argumentId: string;  
};  
  
export { CommentTableEntry };
```

### E.1.149 argupedia-api-master/src/types/recursivePartial.ts

```
export type RecursivePartial<T> = {
  [P in keyof T]?: T[P] extends (infer U)[] ?
    RecursivePartial<U>[]
  : T[P] extends object
  ? RecursivePartial<T[P]>
  : T[P];
};
```

### E.1.150 argupedia-api-master/src/types/users.ts

```
type UserTableEntry = {  
    userId: string;  
    authoredArgumentIds: Array<string>;  
    authoredCommentIds: Array<string>;  
    votedArgumentIds: Array<string>;  
    displayName: string;  
    biography: string;  
    profilePictureURL: string;  
    lastSignedOn: string;  
};  
  
export { UserTableEntry };
```

## E.1.151 argupedia-api-master/tsconfig.json

```
{  
  "extends": "./tsconfig.paths.json",  
  "compilerOptions": {  
    "lib": ["ESNext"],  
    "moduleResolution": "node",  
    "noUnusedLocals": true,  
    "noUnusedParameters": true,  
    "removeComments": true,  
    "sourceMap": true,  
    "target": "ES2020",  
    "outDir": "lib",  
    "esModuleInterop": true  
  },  
  "include": ["src/**/*.ts", "serverless.ts"],  
  "exclude": [  
    "node_modules/**/*",  
    ".serverless/**/*",  
    ".webpack/**/*",  
    ".warmup/**/*",  
    ".vscode/**/*"  
  ],  
  "ts-node": {  
    "require": ["tsconfig-paths/register"]  
  }  
}
```

### E.1.152 argupedia-api-master/tsconfig.paths.json

```
{  
  "compilerOptions": {  
    "baseUrl": ".",  
    "paths": {  
      "@functions/*": ["src/functions/*"],  
      "@libs/*": ["src/libs/*"],  
      "@adapters/*": ["src/adapters/*"],  
      "@helpers/*": ["src/helpers/*"]  
    }  
  }  
}
```

## E.2 Argupedia App Source Code

### E.2.1 argupedia-app-master/capacitor.config.ts

```
import { CapacitorConfig } from "@capacitor/cli";

const config: CapacitorConfig = {
  appId: "io.ionic.starter",
  appName: "argupedia-app",
  webDir: "build",
  bundledWebRuntime: false,
};

export default config;
```

## E.2.2 argupedia-app-master/ionic.config.json

```
{  
  "name": "argupedia-app",  
  "integrations": {  
    "capacitor": {}  
  },  
  "type": "react"  
}
```

### E.2.3 argupedia-app-master/jest.config.js

```
module.exports = {  
  roots: ["<rootDir>/src"],  
  testMatch: [  
    "**/_tests_/**/*.(ts|tsx|js)",  
    "**/?(*.)+(spec|test).+(ts|tsx|js)",  
  ],  
  transform: {  
    "^.+\\.(ts|tsx)$": "ts-jest",  
  },  
  moduleFileExtensions: ["ts", "tsx", "js", "jsx", "json", "node"],  
};
```

## E.2.4 argupedia-app-master/package.json

```
{  
  "name": "argupedia-app",  
  "version": "0.0.1",  
  "private": true,  
  "dependencies": {  
    "@aws-amplify/ui-react": "2.10.3",  
    "@capacitor/app": "1.0.6",  
    "@capacitor/core": "3.3.2",  
    "@capacitor/haptics": "1.1.3",  
    "@capacitor/keyboard": "1.1.3",  
    "@capacitor/status-bar": "1.0.6",  
    "@ionic/react": "6.0.14",  
    "@ionic/react-router": "6.0.14",  
    "@testing-library/jest-dom": "^5.11.9",  
    "@testing-library/react": "^11.2.5",  
    "@testing-library/user-event": "^12.6.3",  
    "@types/jest": "^26.0.20",  
    "@types/lodash": "4.14.181",  
    "@types/node": "^12.19.15",  
    "@types/react": "^16.14.3",  
    "@types/react-dom": "^16.9.10",  
    "@types/react-router": "^5.1.11",  
    "@types/react-router-dom": "^5.1.7",  
    "aws-amplify": "4.3.16",  
    "clsx": "1.1.1",  
    "deepmerge": "4.2.2",  
    "formik": "2.2.9",  
    "ionicons": "^5.4.0",  
    "lodash": "4.17.21",  
    "node-sass": "6.0.1",  
    "react": "18.0.0",  
    "react-dom": "18.0.0",  
    "react-router": "^5.2.0",  
    "react-router-dom": "^5.2.0",  
    "reaflow": "4.2.16",  
    "typescript": "^4.1.3",  
    "web-vitals": "^0.2.4",  
    "workbox-background-sync": "^5.1.4",  
    "workbox-broadcast-update": "^5.1.4",  
    "workbox-cacheable-response": "^5.1.4",  
  }  
}
```

```

"workbox-core": "^5.1.4",
"workbox-expiration": "^5.1.4",
"workbox-google-analytics": "^5.1.4",
"workbox-navigation-preload": "^5.1.4",
"workbox-precaching": "^5.1.4",
"workbox-range-requests": "^5.1.4",
"workbox-routing": "^5.1.4",
"workbox-strategies": "^5.1.4",
"workbox-streams": "^5.1.4",
"zustand": "3.7.1"

},
"scripts": {
  "start": "react-scripts_start",
  "build": "react-scripts_build && mv build www",
  "test": "react-scripts_test --transformIgnorePatterns 'node_modules/(?!(@ionic/react|@ionic/react-router|@ionic/core|@stencil/core|ionicons))'",
  "test:coverage": "react-scripts_test --coverage",
  "eject": "react-scripts_eject",
  "format": "prettier --loglevel warn --write '**/*.{ts,tsx,js,scss}\''",
  "format:check": "prettier --loglevel warn --check '**/*.{ts,tsx,js,scss}\''",
  "lint": "eslint '**/*.ts\'' --fix",
  "lint:check": "eslint '**/*.ts\''"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last_1_chrome_version",
    "last_1_firefox_version",
    "last_1_safari_version"
  ]
},
"devDependencies": {

```

```
"@capacitor/cli": "3.3.2",
"@ionic/prettier-config": "2.0.0",
"prettier": "2.5.0",
"react-scripts": "5.0.0"
},
"description": "An Ionic project"
}
```

## E.2.5 argupedia-app-master/public/manifest.json

```
{  
  "short_name": "Argupedia",  
  "name": "Argupedia",  
  "icons": [  
    {  
      "src": "assets/icon/favicon.png",  
      "sizes": "64x64\u2b3332x32\u2b3324x24\u2b3316x16",  
      "type": "image/x-icon"  
    },  
    {  
      "src": "assets/icon/icon.png",  
      "type": "image/png",  
      "sizes": "512x512",  
      "purpose": "maskable"  
    }  
,  
  ],  
  "start_url": ".",  
  "display": "standalone",  
  "theme_color": "#ffffff",  
  "background_color": "#ffffff"  
}
```

## E.2.6 argupedia-app-master/sonar-project.properties

```
sonar.projectKey=jononon_argupedia-app
sonar.organization=jononon

sonar.sources=src
sonar.exclusions=src/**/*.test.ts, src/**/*.spec.ts, src/**/*.test.tsx, src/**/*.spec.tsx

sonar.tests=src
sonar.test.inclusions=src/**/*.test.ts, src/**/*.spec.ts, src/**/*.test.tsx, src/**/*.spec.tsx
```

## E.2.7 argupedia-app-master/src/App.test.tsx

```
import React from "react";
import { render } from "@testing-library/react";
import App from "./App";

test("renders without crashing", () => {
  const { baseElement } = render(<App />);
  expect(baseElement).toBeDefined();
});
```

## E.2.8 argupedia-app-master/src/App.tsx

```
import { Authenticator } from "@aws-amplify/ui-react";
import "@aws-amplify/ui-react/styles.css";
import {
  IonApp,
  IonContent,
  IonItem,
  IonLabel,
  IonList,
  IonListHeader,
  IonMenu,
  IonMenuToggle,
  IonRouterOutlet,
  IonSplitPane,
  setupIonicReact,
} from "@ionic/react";

import { IonReactRouter } from "@ionic/react-router";
/* Core CSS required for Ionic components to work properly */
import "@ionic/react/css/core.css";
import "@ionic/react/css/display.css";
import "@ionic/react/css/flex-utils.css";
import "@ionic/react/css/float-elements.css";
/* Basic CSS for apps built with Ionic */
import "@ionic/react/css/normalize.css";
/* Optional CSS utils that can be commented out */
import "@ionic/react/css/padding.css";
import "@ionic/react/css/structure.css";
import "@ionic/react/css/text-alignment.css";
import "@ionic/react/css/text-transformation.css";
import "@ionic/react/css/typography.css";
import Amplify from "aws-amplify";
import { Redirect, Route } from "react-router-dom";
import aws_exports from "./aws-exports";
import useStore from "./hooks";
import Argument from "./pages/Argument";
import CreateArgument from "./pages/CreateArgument";
import EditArgument from "./pages/EditArgument";
import EditProfile from "./pages/EditProfile";
import Home from "./pages/Home";
import Profile from "./pages/Profile";
/* Theme variables */
```

```

import "./theme/variables.css";

Amplify.configure(aws_exports);

const App: React.FC = () => {
  setupIonicReact({
    mode: "ios",
  });

  const currentUser = useStore((state) => state.currentUser);

  return (
    <Authenticator /*services={authServices}*/>
      {({ signOut, user }) =>
        <IonApp>
          <IonReactRouter>
            <IonSplitPane contentId="main-content" when={false}>
              <IonMenu contentId="main-content" type="overlay">
                <IonContent>
                  <IonList>
                    <IonListHeader lines="full">
                      {currentUser === null
                        ? ""
                        : `Welcome back, ${currentUser.displayName}`}
                    </IonListHeader>
                    <IonMenuToggle>
                      <IonItem routerDirection="root" routerLink="/">
                        <IonLabel>Home</IonLabel>
                      </IonItem>
                      <IonItem
                        routerDirection="root"
                        routerLink="/argument/create"
                      >
                        <IonLabel>Create New Argument</IonLabel>
                      </IonItem>
                      <IonItem routerDirection="root" routerLink="/profile">
                        <IonLabel>View Your Profile</IonLabel>
                      </IonItem>
                      <IonItem onClick={signOut}>
                        <IonLabel>Sign Out</IonLabel>
                      </IonItem>
                    </IonMenuToggle>
                  </IonList>
                </IonContent>
              </IonMenu>
            </IonSplitPane>
          </IonReactRouter>
        </IonApp>
      }
    </Authenticator>
  );
}

```

```

        </IonList>
    </IonContent>
</IonMenu>
<IonRouterOutlet id="main-content">
    <Route exact path="/profile">
        <Profile />
    </Route>
    <Route exact path="/profile/:userId">
        <Profile />
    </Route>
    <Route exact path="/argument">
        <Argument />
    </Route>
    <Route exact path="/argument/:argumentId">
        <Argument />
    </Route>
    <Route exact path="/argument/:argumentId/edit">
        <EditArgument />
    </Route>
    <Route
        exact
        path="/argument/:argumentId/create/:attackingOrSupporting"
    >
        <CreateArgument />
    </Route>
    <Route exact path="/argument/create">
        <CreateArgument />
    </Route>
    <Route exact path="/profile/edit">
        <EditProfile />
    </Route>
    <Route exact path="/home">
        <Home />
    </Route>
    <Route exact path="/">
        <Redirect to="/home" />
    </Route>
</IonRouterOutlet>
</IonSplitPane>
</IonReactRouter>
</IonApp>
)}

```

```
</Authenticator>
);
};

export default App;
```

## E.2.9 argupedia-app-master/src/aws-exports.js

```
/* eslint-disable */
// WARNING: DO NOT EDIT. This file is automatically generated by AWS Amplify. It will be overwritten.

const awsmobile = {
  aws_project_region: "eu-west-2",
  aws_cognito_region: "eu-west-2",
  aws_user_pools_id: "eu-west-2_oB09hMVAN",
  aws_user_pools_web_client_id: "112n7hvt0o9c6cb180memvqpla",
  oauth: {},
  aws_cognito_username_attributes: ["EMAIL"],
  aws_cognito_social_providers: [],
  aws_cognito_signup_attributes: [],
  aws_cognito_mfa_configuration: "OFF",
  aws_cognito_mfa_types: [],
  aws_cognito_password_protection_settings: {
    passwordPolicyMinLength: 8,
    passwordPolicyCharacters: [
      "REQUIRES_LOWERCASE",
      "REQUIRES_UPPERCASE",
      "REQUIRES_NUMBERS",
      "REQUIRES_SYMBOLS",
    ],
  },
  aws_cognito_verification_mechanisms: ["EMAIL"],
};

export default awsmobile;
```

## E.2.10 argupedia-app-master/src/components/ArgumentCard.css

```
.container {  
    text-align: center;  
    position: absolute;  
    left: 0;  
    right: 0;  
    top: 50%;  
    transform: translateY(-50%);  
}  
  
.container strong {  
    font-size: 20px;  
    line-height: 26px;  
}  
  
.container p {  
    font-size: 16px;  
    line-height: 22px;  
    color: #8c8c8c;  
    margin: 0;  
}  
  
.container a {  
    text-decoration: none;  
}
```

## E.2.11 argupedia-app-master/src/components/ArgumentCard.tsx

```
import {
  IonCard,
  IonCardContent,
  IonCardHeader,
  IonCardSubtitle,
} from "@ionic/react";
import "./ArgumentCard.css";

interface CardProps {
  argumentText?: string;
  argumentId?: string;
}

const ArgumentCard: React.FC<CardProps> = ({ argumentText, argumentId }) => {
  return (
    <IonCard routerLink={`/argument/${argumentId}`}>
      <IonCardHeader>
        <IonCardSubtitle>Argument</IonCardSubtitle>
      </IonCardHeader>
      <IonCardContent>{argumentText}</IonCardContent>
    </IonCard>
  );
};

export default ArgumentCard;
```

## E.2.12 argupedia-app-master/src/components/Box/Box.tsx

```
type BoxProps = {
  className?: string;
  m?: number; // Margin.
  my?: number; // Margin vertical.
  mx?: number; // Margin horizontal.
  p?: number; // Padding.
  py?: number; // Padding vertical.
  px?: number; // Padding horizontal.
};

const Box: React.FC<BoxProps> = ({
  className,
  children,
  m,
  my,
  mx,
  p,
  py,
  px,
}) => {
  const style = {
    marginTop: m ?? my,
    marginBottom: m ?? my,
    marginLeft: m ?? mx,
    marginRight: m ?? mx,
    paddingTop: p ?? py,
    paddingBottom: p ?? py,
    paddingLeft: p ?? px,
    paddingRight: p ?? px,
  };
  return (
    <div className={className} style={style}>
      {children}
    </div>
  );
};

export default Box;
```

### E.2.13 argupedia-app-master/src/components/Box/index.ts

```
export { default } from "./Box";
```

## E.2.14 argupedia-app-master/src/components/CommentForm.tsx

```
import {
  IonButton,
  IonItem,
  IonLabel,
  IonList,
  IonTextarea,
} from "@ionic/react";

import { Field, FieldProps, Form, Formik } from "formik";
import api from "../services";
import "./ArgumentCard.css";
import Box from "./Box";

interface CardProps {
  argumentId: string;
}

const initialValues = {
  commentBody: "",
};

const CommentForm: React.FC<CardProps> = ({ argumentId }) => {
  const IonTextareaWrapper = ({ field, form, ...props }: FieldProps & { label: string; name: string; test: string }) => (
    <IonItem>
      <IonLabel position="stacked">
        <b>{props.label}</b>
      </IonLabel>
      <IonTextarea
        onChange={(e) => form.setFieldValue(field.name, e.detail.value)}
        {...props}
        {...field}
      ></IonTextarea>
    </IonItem>
  );
  return (
    <>
```

```

<Box py={10} />
<Formik<typeof initialValues>
  initialValues={initialValues}
  onSubmit={(values) => {
    console.log(values);
    api
      .createCommentForArgumentWithId(argumentId, values.commentBody)
      .then(() => {
        window.location.reload();
      });
  }};
>
{({ submitForm, touched, values }) => (
  <Form>
    <IonList>
      <Field
        name={"commentBody"}
        label={"Add a comment"}
        type="text"
        component={IonTextareaWrapper}
      />
      <Box px={20}>
        <IonButton onClick={submitForm}>Post</IonButton>
      </Box>
    </IonList>
  </Form>
)}
</Formik>
</>
);
};

export default CommentForm;

```

## E.2.15 argupedia-app-master/src/components/Toolbar.tsx

```
import {
  IonButton,
  IonButtons,
  IonHeader,
  IonMenuButton,
  IonTitle,
  IonToolbar,
} from "@ionic/react";

import { Auth } from "aws-amplify";
import { useEffect, useState } from "react";
import useStore from "../hooks";
import api from "../services";
import { UserTableEntry } from "../types";

const Toolbar: React.FC = () => {
  const storeUser = useStore((state) => state.storeUser);
  const currentUser = useStore((state) => state.currentUser);
  const [userDetails, setUserDetails] = useState<UserTableEntry>();

  useEffect(() => {
    async function getUserDetails() {
      const cognitoSession = await Auth.currentSession();
      const userDetails = await api.getUserById(
        cognitoSession.getIdToken().payload.sub
      );
      storeUser(userDetails);
      return userDetails;
    }

    async function createUser() {
      storeUser(await api.createUser());
    }

    getUserDetails().catch(() => createUser());
  }, []);

  return (
    <>
      <IonHeader>
        <IonToolbar>
```

```
<IonButtons slot="start">
  <IonMenuButton autoHide={false}></IonMenuButton>
</IonButtons>
<IonTitle>Argupedia</IonTitle>
</IonToolbar>
</IonHeader>
</>
);
};

export default Toolbar;
```

## E.2.16 argupedia-app-master/src/constants/argumentTypePlaceholders.ts

```
const argumentTypePlaceholders = {
```

```
  ActionArgument: {
```

```
    circumstance: "there\uis\uapandemic",
    action: "full\ulockdown",
    newCircumstance: "the\uapandemic\uends",
    goal: "less\udeath",
    value: "the\uhealth\uof\uthe\upopulation",
  },
```

```
  ArgumentFromPositionToKnow: {
```

```
    person: "the\udirector\uof\uthe\uNHS",
    fact: "there\uis\uapandemic",
    proposition: "true",
  },
```

```
  AppealToExpertOpinion: {
```

```
    expert: "Dr.\uSmith",
    domain: "virology",
    fact: "there\uis\uapandemic",
    proposition: "true",
  },
```

```
  AppealToPopularOpinion: {
```

```
    fact: "there\uis\uapandemic",
    proposition: "true",
  },
```

```
  ArgumentFromAnalogy: {
```

```
    originalCase: "Hong\uKong",
    similarCase: "United\uKingdom",
    fact: "social\u distancing\u reduced\u the\u impact\u of\u covid",
    proposition: "true",
  },
```

```
  ArgumentFromCorrelationToCause: {
```

```
    cause: "inflation",
    effect: "higher\u petrol\u prices",
  },
```

```
  ArgumentFromPositiveOrNegativeConsequences: {
```

```

cause: "more\u00a5money\u00a5supply",
effect: "higher\u00a5petrol\u00a5prices",
consequence: "bad",
},

SlipperySlopeArgument: {
firstStepPremise: "working\u00a5out\u00a5more",
recursivePremise: "happier\u00a5emotions",
consequence: "good",
},

ArgumentFromSign: {
finding: "the\u00a5birds\u00a5are\u00a5flying\u00a5low",
fact: "it\u00a5is\u00a5going\u00a5to\u00a5rain",
proposition: "true",
},

ArgumentFromCommitment: {
entity: "the\u00a5director\u00a5of\u00a5the\u00a5NHS",
existingCommitment: "wearing\u00a5face\u00a5masks\u00a5in\u00a5public",
newCommitment: "hand\u00a5sanitizer\u00a5stations\u00a5in\u00a5tube\u00a5stations",
},

ArgumentFromInconsistentCommitment: {
entity: "the\u00a5director\u00a5of\u00a5the\u00a5NHS",
commitment: "wearing\u00a5face\u00a5masks\u00a5in\u00a5public",
otherEvidence: "photos\u00a5of\u00a5the\u00a5director\u00a5without\u00a5her\u00a5mask",
},

DirectAdHominemArgument: {
entity: "Norman\u00a5Bates",
criticism: "he\u00a5is\u00a5a\u00a5person\u00a5of\u00a5bad\u00a5character",
},

CircumstantialAdHominemArgument: {
entity: "the\u00a5director\u00a5of\u00a5the\u00a5NHS",
originalArgument:
"wearing\u00a5face\u00a5masks\u00a5in\u00a5public\u00a5will\u00a5reduce\u00a5the\u00a5spread\u00a5of\u00a5covid",
otherEvidence: "photos\u00a5of\u00a5the\u00a5director\u00a5without\u00a5her\u00a5mask",
},

ArgumentFromVerbalClassification: {

```

```
entity: "Boris\u201cJohnson",
property: "English",
classification: "puts\u201c\u00e9milk\u201c\u00e9n\u201c\u00e9s\u201c\u00e9tea",
},
};

export { argumentTypePlaceholders };
```

## E.2.17 argupedia-app-master/src/constants/argumentTypeSchema.ts

```
type ArgumentTypeSchema = {  
  [key: string]: {  
    [key: string]: string;  
  };  
};
```

```
const argumentTypeSchema: ArgumentTypeSchema = {
```

```
  ActionArgument: {  
    circumstance: "string",  
    action: "string",  
    newCircumstance: "string",  
    goal: "string",  
    value: "string",  
  },
```

```
  ArgumentFromPositionToKnow: {  
    person: "string",  
    fact: "string",  
    proposition: "boolean",  
  },
```

```
  AppealToExpertOpinion: {  
    expert: "string",  
    domain: "string",  
    fact: "string",  
    proposition: "boolean",  
  },
```

```
  AppealToPopularOpinion: {  
    fact: "string",  
    proposition: "boolean",  
  },
```

```
  ArgumentFromAnalogy: {  
    originalCase: "string",  
    similarCase: "string",  
    fact: "string",  
    proposition: "boolean",  
  },
```

```

ArgumentFromCorrelationToCause: {
  cause: "string",
  effect: "string",
},

ArgumentFromPositiveOrNegativeConsequences: {
  cause: "string",
  effect: "string",
  consequence: "consequenceType",
},

SlipperySlopeArgument: {
  firstStepPremise: "string",
  recursivePremise: "string",
  consequence: "consequenceType",
},

ArgumentFromSign: {
  finding: "string",
  fact: "string",
  proposition: "boolean",
},

ArgumentFromCommitment: {
  entity: "string",
  existingCommitment: "string",
  newCommitment: "string",
},

ArgumentFromInconsistentCommitment: {
  entity: "string",
  commitment: "string",
  otherEvidence: "string",
},

DirectAdHominemArgument: {
  entity: "string",
  criticism: "string",
},

CircumstantialAdHominemArgument: {
  entity: "string",
}

```

```
originalArgument: "string",
otherEvidence: "string",
},
ArgumentFromVerbalClassification: {
entity: "string",
property: "string",
classification: "string",
},
};

export { argumentTypeSchema };
```

## E.2.18 argupedia-app-master/src/constants/criticalQuestions.ts

```

type CriticalQuestions = {
  [key: string]: Array<string>;
};

const criticalQuestions: CriticalQuestions = {

  ActionArgument: [
    "Is_it_true_that_$circumstance?",
    "Assuming_$circumstance,_does_$action_bring_about_$newCircumstance?",
    "Assuming_the_$circumstance_and_that_$action_will_bring_about_$newCircumstance,_will_$action_bring_about_$goal?",
    "Does_$goal_realise_$value?",
    "Are_there_alternative_ways_of_realising_$newCircumstance?",
    "Are_there_alternative_ways_of_realising_$goal?",
    "Are_there_alternative_ways_of_promoting_$value?",
    "Does_doing_$action_have_a_side_effect_which_demotes_$value?",
    "Does_doing_$action_have_a_side_effect_which_demotes_some_other_value?",
    "Does_doing_$action_promote_some_other_value?",
    "Does_doing_$action_preclude_some_other_action_which_would_promote_some_other_value?",
    "Is_it_possible_that_$circumstance?",
    "Is_the_$action_possible?",
    "Is_$newCircumstance_possible?",
    "Can_$goal_be_realised?",
    "Is_$value_indeed_a_legitimate_value?",

  ],
  ArgumentFromPositionToKnow: [
    "Is_$person_really_in_a_position_to_know_whether_$fact_is_true?_What_is_it_about_$person_that_makes_them_likely_to_know_$fact?",

    "Is_$person_an_honest,_trustworthy,_and_reliable_source?_Would_$person_have_any_reason_to_mislead?",

    "Did_$person_really_assert_that_$fact_is_$proposition?_Are_we_hearing_what_$person_said_first-hand_or_second-hand?_Is_there_reason_to_be_suspicious_about_the_fidelity_of_the_information-transfer?",

  ],
  AppealToExpertOpinion: [
    "How_credible_is_$expert_as_an_expert?",

    "Is_$expert_an_expert_in_the_field_that_$fact_is_in?",

    "What_did_$expert_assert_that_implies_$fact?",

    "Is_$expert_personally_reliable_and_trustworthy?_Do_we_have_any_reason_to_think_$expert_is_less_than_honest?",

    "Is_$fact_consistent_with_what_other_experts_assert?",

  ],
}

```

```

    "Is the expert's evidence based on evidence?",  

],  
  

AppealToPopularOpinion: [  

    "What evidence do we have for believing that the fact is generally accepted?",  

    "Even if the fact is generally accepted as being true, are there good reasons for doubting its veracity?",  

],  
  

ArgumentFromAnalogy: [  

    "Is the proposition in the original case?",  

    "Are there differences between the original case and the similar case that would tend to undermine the force of the similarity cited?",  

    "Is there some other case that is also similar to the original case but in which the fact is the proposition opposite?",  

],  
  

ArgumentFromCorrelationToCause: [  

    "Is there really a correlation between cause and effect?",  

    "Is there any reason for thinking the correlation is anything more than a coincidence?",  

    "Could there be some third factor that is causing both cause and effect?",  

],  
  

ArgumentFromPositiveOrNegativeConsequences: [  

    "How strong is the probability or plausibility that the effect will occur?",  

    "What evidence supports the claim that the effect will occur?",  

    "Are there consequences of not doing the cause that ought to be taken into account?",  

    "Is the cause really a consequence?",  

],  
  

SlipperySlopeArgument: [  

    "What intervening propositions in the sequence linking the first step premise to the recursive premise are actually given?",  

    "What other steps are required to fill in the sequence to make it plausible?",  

    "What are the weakest links in the sequence, the places where key critical questions need to be asked?",  

],  
  

ArgumentFromSign: [  

    "What is the strength of the correlation between the finding and the fact?",  

    "Are there other events that would more reliably account for the finding?",  

],  
  

ArgumentFromCommitment: [  

    "What evidence are we relying on to claim that the entity is committed to the existing commitment?",  

]

```

```

    "Is there room for questioning whether there is an exception in this case to the general rule that commitment
    to $existingCommitment implies commitment to $newCommitment?",

],


ArgumentFromInconsistentCommitment: [
    "What is the evidence supposedly showing that $entity is committed to $commitment?",

    "What further evidence shows that $person is not really committed to $commitment?",

    "How does the evidence show that there is a conflict of commitments?",

],


DirectAdHominemArgument: [
    "How well supported by evidence is $criticism?",

    "Is $criticism relevant in the dialogue in which the argument is made?",

    "Is the conclusion that $entity's argument should not be accepted absolute or merely one factor to consider
    when assessing the argument?",

],


CircumstantialAdHominemArgument: [
    "Is $otherEvidence reliable evidence?",

    "Could the practical inconsistency be resolved by further dialog? Should $entity be given a chance to explain
    themselves?",

    "Is character a relevant consideration in this argumentative dialogue?",

    "What are we entitled to conclude? Can we ignore $originalArgument? Or are we only entitled to ignore $entity
    's contributions to a more general debate?",

],


ArgumentFromVerbalClassification: [
    "What evidence is there for thinking that $entity definitely is $property? Are there reasons for thinking
    $entity belongs to another category of objects?",

    "Is saying that those who are $property probably $classification a stipulative or biased definition that is
    subject to doubt?",

],


};

export { criticalQuestions };

```

## E.2.19 argupedia-app-master/src/helpers/argupedia/argumentTranslations.ts

```
import {  
    ActionArgument,  
    AppealToExpertOpinion,  
    AppealToPopularOpinion,  
    Argument,  
    ArgumentFromAnalogy,  
    ArgumentFromCommitment,  
    ArgumentFromCorrelationToCause,  
    ArgumentFromInconsistentCommitment,  
    ArgumentFromPositionToKnow,  
    ArgumentFromPositiveOrNegativeConsequences,  
    ArgumentFromSign,  
    ArgumentFromVerbalClassification,  
    CircumstantialAdHominemArgument,  
    DirectAdHominemArgument,  
    SlipperySlopeArgument,  
} from "../../types/arguments";  
  
import { capitalizeFirstLetter } from "./capitalizeFirstLetter";  
  
const argumentTranslations = (argumentType: string, argument: Argument) => {  
    switch (argumentType) {  
        case "ActionArgument": {  
            const actionArgument = argument as ActionArgument;  
  
            return `In the current circumstance ${actionArgument.circumstance}, we should perform action ${  
                actionArgument.action} which will result in new circumstances ${actionArgument.newCircumstance} which  
                will realise new goal ${actionArgument.goal} which will promote value ${actionArgument.value}.`;  
        }  
        case "ArgumentFromPositionToKnow": {  
            const argumentFromPositionToKnow = argument as ArgumentFromPositionToKnow;  
  
            return `${capitalizeFirstLetter(  
                argumentFromPositionToKnow.person  
            )} is in a position to know whether ${  
                argumentFromPositionToKnow.fact  
            } is true or false and they suggest that it is ${  
                argumentFromPositionToKnow.proposition  
            }.`;  
        }  
        case "AppealToExpertOpinion": {  
    }
```

```

const appealToExpertOpinion = argument as AppealToExpertOpinion;

return `${appealToExpertOpinion.expert} is an expert in ${appealToExpertOpinion.domain} and suggests that
it is ${appealToExpertOpinion.proposition} that ${appealToExpertOpinion.fact}.`;

}

case "AppealToPopularOpinion": {
    const appealToPopularOpinion = argument as AppealToPopularOpinion;

    return `${capitalizeFirstLetter(
        appealToPopularOpinion.fact
    )} is generally accepted as being ${appealToPopularOpinion.proposition}.`;
}

case "ArgumentFromAnalogy": {
    const argumentFromAnalogy = argument as ArgumentFromAnalogy;

    return `Generally, ${argumentFromAnalogy.originalCase} is similar to ${
        argumentFromAnalogy.similarCase
    }. ${capitalizeFirstLetter(argumentFromAnalogy.fact)} is ${
        argumentFromAnalogy.proposition
    } in ${argumentFromAnalogy.originalCase} so it is probably also ${
        argumentFromAnalogy.proposition
    } in ${argumentFromAnalogy.similarCase}.`;
}

case "ArgumentFromCorrelationToCause": {
    const argumentFromCorrelationToCause =
        argument as ArgumentFromCorrelationToCause;

    return `${capitalizeFirstLetter(
        argumentFromCorrelationToCause.cause
    )} cause ${argumentFromCorrelationToCause.effect}.`;
}

case "ArgumentFromPositiveOrNegativeConsequences": {
    const argumentFromPositiveOrNegativeConsequences =
        argument as ArgumentFromPositiveOrNegativeConsequences;

    const consequence =
        argumentFromPositiveOrNegativeConsequences.consequence === undefined
            ? ""
            : argumentFromPositiveOrNegativeConsequences.consequence.toLowerCase();

    return `If ${argumentFromPositiveOrNegativeConsequences.cause} is brought about, the consequences will be ${
        argumentFromPositiveOrNegativeConsequences.effect}. This is ${consequence}.`;
}

```

```

}

case "SlipperySlopeArgument": {
  const slipperySlopeArgument = argument as SlipperySlopeArgument;
  const consequence =
    slipperySlopeArgument.consequence === undefined
      ? ""
      : slipperySlopeArgument.consequence.toLowerCase();
  return `${capitalizeFirstLetter(
    slipperySlopeArgument.firstStepPremise
  )} is up for consideration but doing so will lead to ${
    slipperySlopeArgument.recursivePremise
  }. This is a ${consequence} outcome.`;
}

case "ArgumentFromSign": {
  const argumentFromSign = argument as ArgumentFromSign;

  return `${capitalizeFirstLetter(
    argumentFromSign.finding
  )} is true in this situation. ${capitalizeFirstLetter(
    argumentFromSign.fact
  )} is generally indicated as true when ${
    argumentFromSign.finding
  } is true. Therefore ${argumentFromSign.fact} is probably ${
    argumentFromSign.proposition
  }.`;
}

case "ArgumentFromCommitment": {
  const argumentFromCommitment = argument as ArgumentFromCommitment;

  return `${capitalizeFirstLetter(
    argumentFromCommitment.entity
  )} is committed to ${
    argumentFromCommitment.existingCommitment
  }. Generally when someone is committed to ${
    argumentFromCommitment.existingCommitment
  }, they are also committed to ${
    argumentFromCommitment.newCommitment
  }. Therefore, ${argumentFromCommitment.entity} is probably committed to ${
    argumentFromCommitment.newCommitment
  }.`;
}

case "ArgumentFromInconsistentCommitment": {

```

```

const argumentFromInconsistentCommitment =
    argument as ArgumentFromInconsistentCommitment;

return `${capitalizeFirstLetter(
    argumentFromInconsistentCommitment.entity
)} is indicated that they are committed to ${
    argumentFromInconsistentCommitment.commitment
}. However, ${
    argumentFromInconsistentCommitment.otherEvidence
} demonstrates that ${
    argumentFromInconsistentCommitment.entity
} is not really committed to ${
    argumentFromInconsistentCommitment.commitment
}. Therefore, ${
    argumentFromInconsistentCommitment.entity
}` +
    `'s commitments are inconsistent.`;
}

case "DirectAdHominemArgument": {
    const directAdHominemArgument = argument as DirectAdHominemArgument;

    return `${capitalizeFirstLetter(
        directAdHominemArgument.entity
)}'s comments should not be accepted because ${
        directAdHominemArgument.criticism
}.`;
}

case "CircumstantialAdHominemArgument": {
    const circumstantialAdHominemArgument =
        argument as CircumstantialAdHominemArgument;

    return `${capitalizeFirstLetter(
        circumstantialAdHominemArgument.entity
)}'s is an advocate of this ${
        circumstantialAdHominemArgument.originalArgument
}, but ${
        circumstantialAdHominemArgument.otherEvidence
} suggests that ${
        circumstantialAdHominemArgument.entity
} is actually not committed to this argument.`;
}

case "ArgumentFromVerbalClassification": {
    const argumentFromVerbalClassification =
        argument as ArgumentFromVerbalClassification;
}

```

```
    return `${capitalizeFirstLetter(  
      argumentFromVerbalClassification.entity  
    )} is ${argumentFromVerbalClassification.property}. Therefore ${  
      argumentFromVerbalClassification.entity  
    } probably ${argumentFromVerbalClassification.classification}.`;  
  }  
  
  default:  
    throw new Error("Invalid argument type");  
  }  
  
};  
  
export { argumentTranslations };
```

## E.2.20 argupedia-app-master/src/helpers/argupedia/capitalizeFirstLetter.ts

```
const capitalizeFirstLetter = (str: string) => {
  if (str === undefined) {
    return undefined;
  }
  return str.charAt(0).toUpperCase() + str.slice(1);
};

export { capitalizeFirstLetter };
```

## E.2.21 argupedia-app-master/src/helpers/argupedia/formatCriticalQuestions.ts

```
import { argumentTypeSchema } from "../../constants/argumentTypeSchema";
import { criticalQuestions } from "../../constants/criticalQuestions";
import { Argument } from "../../types/arguments";

const formatCriticalQuestions = (argumentType: string, argument: Argument) =>
  criticalQuestions[argumentType].map((question) => {
    for (const key of Object.keys(argumentTypeSchema[argumentType]))) {
      question = question.replaceAll(`$$${key}`, (argument as any)[key]);
    }
    return question;
  });

export { formatCriticalQuestions };
```

## E.2.22 argupedia-app-master/src/helpers/argupedia/pascalCaseToSentence.ts

//source: <https://stackoverflow.com/questions/26188882/split-pascal-case-in-javascript-certain-case>

```
const pascalCaseToSentence = (str: string) => {
  return (
    str
    // Look for long acronyms and filter out the last letter
    .replace(/([A-Z]+)([A-Z][a-z])/g, "$1 $2")
    // Look for lower-case letters followed by upper-case letters
    .replace(/([a-z\d])([A-Z])/g, "$1 $2")
    // Look for lower-case letters followed by numbers
    .replace(/([a-zA-Z])(\d)/g, "$1 $2")
    .replace(/\./, function (str) {
      return str.toUpperCase();
    })
    // Remove any white space left around the word
    .trim()
  );
};

export { pascalCaseToSentence };
```

## E.2.23 argupedia-app-master/src/helpers/argupedia/timeDifference.ts

```
const timeDifference = (options: {
  current?: Date;
  previous: Date;
}): string => {
  const { current, previous } = options;

  var msPerMinute = 60 * 1000;
  var msPerHour = msPerMinute * 60;
  var msPerDay = msPerHour * 24;
  var msPerMonth = msPerDay * 30;
  var msPerYear = msPerDay * 365;

  var elapsed = (current || new Date()).getTime() - previous.getTime();

  if (elapsed < msPerMinute) {
    return Math.round(elapsed / 1000) + " seconds ago";
  } else if (elapsed < msPerHour) {
    return Math.round(elapsed / msPerMinute) + " minutes ago";
  } else if (elapsed < msPerDay) {
    return Math.round(elapsed / msPerHour) + " hours ago";
  } else if (elapsed < msPerMonth) {
    return "about " + Math.round(elapsed / msPerDay) + " days ago";
  } else if (elapsed < msPerYear) {
    return "about " + Math.round(elapsed / msPerMonth) + " months ago";
  } else {
    return "about " + Math.round(elapsed / msPerYear) + " years ago";
  }
};

export { timeDifference };
```

## E.2.24 argupedia-app-master/src/helpers/mergeOptions/index.ts

```
export * from "./mergeOptions";
```

## E.2.25 argupedia-app-master/src/helpers/mergeOptions/mergeOptions.ts

```
import merge from "deepmerge";

const mergeOptions: merge.Options = {
  arrayMerge: (_destinationArray, sourceArray, _options) => sourceArray,
};

export { mergeOptions };
```

## E.2.26 argupedia-app-master/src/hooks/index.ts

```
export { default } from "./useStore";
```

## E.2.27 argupedia-app-master/src/hooks/useStore.ts

```
import { mergeOptions } from "../helpers/mergeOptions";
import create from "zustand";
import merge from "deepmerge";
import { UserTableEntry } from "../types";

type AppState = {
  isAuthenticated: boolean;
  currentUser: UserTableEntry | null;
  storeUser: (user: UserTableEntry | null) => void;
};

const useStore = create<AppState>((set) => ({
  isAuthenticated: false,
  currentUser: null,
  storeUser: (currentUser: UserTableEntry | null) =>
    set((state) => merge(state, { currentUser }, mergeOptions), true),
}))();

export default useStore;
```

## E.2.28 argupedia-app-master/src/index.tsx

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
import * as serviceWorkerRegistration from "./serviceWorkerRegistration";
import reportWebVitals from "./reportWebVitals";

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
);

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://cra.link/PWA
serviceWorkerRegistration.unregister();

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

**E.2.29 argupedia-app-master/src/pages/Argument.css**

## E.2.30 argupedia-app-master/src/pages/Argument.tsx

```
import {
  IonButton,
  IonCard,
  IonCardContent,
  IonCardHeader,
  IonCardSubtitle,
  IonCardTitle,
  IonCol,
  IonContent,
  IonGrid,
  IonItem,
  IonLabel,
  IonList,
  IonPage,
  IonRouterLink,
  IonRow,
  IonText,
} from "@ionic/react";

import { Auth } from "aws-amplify";
import React, { useEffect, useState } from "react";
import { useHistory, useParams } from "react-router";
import {
  Canvas,
  Edge,
  EdgeData,
  EdgeProps,
  Node,
  NodeData,
  NodeProps,
} from "reaflow";

import CommentForm from "../components/CommentForm";
import Toolbar from "../components/Toolbar";
import { argumentTranslations } from "../helpers/argupedia/argumentTranslations";
import { capitalizeFirstLetter } from "../helpers/argupedia/capitalizeFirstLetter";
import { pascalCaseToSentence } from "../helpers/argupedia/pascalCaseToSentence";
import { timeDifference } from "../helpers/argupedia/timeDifference";
import useStore from "../hooks";
import api from "../services";
import { UserTableEntry } from "../types";
import { ArgumentTableEntry } from "../types/arguments";
```

```

import { ArgumentComments } from "../types/comments";
import "./Profile.css";

const Argument: React.FC = () => {
  const storeUser = useStore((state) => state.storeUser);
  const currentUser = useStore((state) => state.currentUser);
  const [currentArgument, setCurrentArgument] = useState<ArgumentTableEntry>();
  const [currentArgumentAuthor, setCurrentArgumentAuthor] =
    useState<UserTableEntry>();
  const [nodes, setNodes] = useState<Array<NodeData>>([]);
  const [edges, setEdges] = useState<Array<EdgeData>>([]);
  const [argumentComments, setArgumentComments] = useState<
    Array<ArgumentComments>
  >([]);

  const history = useHistory();

  const { argumentId } = useParams<{ argumentId: string }>();

  useEffect(() => {
    async function getUserDetails() {
      const cognitoSession = await Auth.currentSession();
      const userDetails = await api.getUserById(
        cognitoSession.getIdToken().payload.sub
      );
      storeUser(userDetails);
      return userDetails;
    }

    async function createUser() {
      storeUser(await api.createUser());
    }

    getUserDetails().catch(() => createUser());
  }, []);

  useEffect(() => {
    async function getArgumentGraphForArgumentWithId(argumentId: string) {
      const argumentGraphEntries = await api.getArgumentGraphForArgumentWithId(
        argumentId
      );
      console.log(argumentGraphEntries);
    }
  });
}

```

```

const argumentParents: { [key: string]: string } = {};

for (const argumentTableEntry of argumentGraphEntries.filter(
  (argumentTableEntry) =>
    argumentTableEntry.supportsArgumentId !== undefined
)) {
  argumentParents[argumentTableEntry.argumentId] =
    argumentTableEntry.supportsArgumentId as string;
}

const getHeadOfSupportingArgumentTree = (
  argumentId: string,
  argumentParents: { [key: string]: string }
): string => {
  if (argumentParents[argumentId] === undefined) {
    return argumentId;
  } else {
    return getHeadOfSupportingArgumentTree(
      argumentParents[argumentId],
      argumentParents
    );
  }
};

const argumentParentsSet = new Set<string>();

const nodes: Array<NodeData> = argumentGraphEntries.map(
  (argumentGraphEntry) => {
    return {
      id: argumentGraphEntry.argumentId,
      text:
        argumentGraphEntry.argumentText ||
        argumentTranslations(
          argumentGraphEntry.argumentType,
          argumentGraphEntry.argument
        ),
      height: 100,
      width: 250,
      data: {
        label: argumentGraphEntry.label,
        isSelected: argumentGraphEntry.argumentId === argumentId,
      }
    };
  }
);

```

```

        isParentNode: false,
    },
},
);
);

setNodes(nodes);

setEdges(
argumentGraphEntries
.filter(
(argumentGraphEntry) =>
argumentGraphEntry.attacksArgumentId !== undefined ||
argumentGraphEntry.supportsArgumentId !== undefined
)
.map((argumentGraphEntry) => {
if (argumentGraphEntry.attacksArgumentId !== undefined) {
return {
id:
argumentGraphEntry.argumentId +
"-" +
argumentGraphEntry.attacksArgumentId,
from: argumentGraphEntry.argumentId,
to: argumentGraphEntry.attacksArgumentId,
isSupporting: false,
};
} else if (argumentGraphEntry.supportsArgumentId !== undefined) {
return {
id:
argumentGraphEntry.argumentId +
"-" +
argumentGraphEntry.supportsArgumentId,
from: argumentGraphEntry.argumentId,
to: argumentGraphEntry.supportsArgumentId,
isSupporting: true,
};
} else {
return {
id: "",
from: "",
to: "",
isSupporting: false,
};
}
});
);

```

```

        );
    }
}

);

};

const getCurrentArgument = async () => {
    const argument = await api.getArgumentWithId(argumentId);
    setCurrentArgument(argument);
    await api
        .getUserById(argument.authorId)
        .then((user) => {
            setCurrentArgumentAuthor(user);
        })
        .catch(() => {
            setCurrentArgumentAuthor(undefined);
        });
};

const getCommentsForArgumentWithId = async () => {
    setArgumentComments(await api.getCommentsForArgumentWithId(argumentId));
};

if (argumentId !== undefined) {
    getArgumentGraphForArgumentWithId(argumentId);
    getCurrentArgument();
    getCommentsForArgumentWithId();
}
}, [argumentId]);

const deleteComment = async (commentId: string) => {
    await api.deleteCommentWithId(commentId);
    if (currentArgument !== undefined) {
        setArgumentComments(
            await api.getCommentsForArgumentWithId(currentArgument.argumentId)
        );
    }
};

return (
<IonPage>
<Toolbar />

```

```

<IonContent>
  <IonGrid>
    <IonRow>
      <IonCol size="6">
        <IonCard>
          <div style={{ position: "relative", height: "600px" }}>
            {nodes.length !== 0 ? (
              <Canvas
                nodes={nodes}
                edges={edges}
                fit={true}
                direction={"UP"}
                node={({node: NodeProps}) => {
                  if (node.properties.data.isParentNode === true) {
                    return <Node {...node} />;
                  }
                  return (
                    <Node
                      {...node}
                      style={{
                        fill:
                          node.properties.data.label === "IN"
                            ? "green"
                            : "red",
                        stroke: node.properties.data.isSelected
                            ? "black"
                            : "lightgray",
                      }}
                      onClick={() => {
                        history.push(`/${argument}/${node.properties.id}`);
                        window.location.reload();
                      }}
                    />
                  );
                }};
                edge={({edge: EdgeProps}) => {
                  const edgeProperties = edge.properties as any;
                  const isSupporting =
                    edgeProperties === undefined
                      ? false
                      : edgeProperties.isSupporting;
                  return (

```

```

<Edge
  {...edge}
  style={{
    stroke: isSupporting ? "green" : "red",
  }}
/>
);
}

/>
) : (
// <IonText>Loading...</IonText>
<IonText>Loading...</IonText>
)
}

</div>
</IonCard>
</IonCol>
<IonCol>
<IonCard>
{currentArgument === undefined ? (
<IonText>Loading...</IonText>
) : (
<>
<IonCardHeader>
<IonCardSubtitle>Selected Argument</IonCardSubtitle>
<IonCardTitle>
{currentArgument.argumentText ||
argumentTranslations(
  currentArgument.argumentType,
  currentArgument.argument
)}
</IonCardTitle>
</IonCardHeader>
<IonCardContent>
<IonGrid>
<IonRow>
<IonCol>
<p>Argument Type</p>
</IonCol>
<IonCol>
<p>
{pascalCaseToSentence(
  currentArgument.argumentType
)
}
</IonCol>

```

```

        )}
      </p>
    </IonCol>
  </IonRow>
  {Object.keys(currentArgument.argument).map(
    (key, index) => (
      <IonRow key={index}>
        <IonCol>
          <p>{capitalizeFirstLetter(key)}</p>
        </IonCol>
        <IonCol>
          <p>
            {pascalCaseToSentence(
              (currentArgument.argument as any)[key]
            )}
          </p>
        </IonCol>
      </IonRow>
    )
  )}
<IonRow>
  <IonCol>
    <p>Label</p>
  </IonCol>
  <IonCol>
    <p>{pascalCaseToSentence(currentArgument.label)}</p>
  </IonCol>
</IonRow>
<IonRow>
  <IonCol>
    <p>Author</p>
  </IonCol>
  <IonCol>
    {currentArgumentAuthor === undefined ? (
      <p>[deleted]</p>
    ) : (
      <IonRouterLink
        routerLink={`/profile/${currentArgumentAuthor.userId}`}
      >
        <p>{currentArgumentAuthor.displayName}</p>
      </IonRouterLink>
    )}
  
```

```

        </IonCol>
    </IonRow>
    <IonRow>
        <IonCol>
            <p>Last Modified</p>
        </IonCol>
        <IonCol>
            <p>
                {timeDifference({
                    previous: new Date(
                        currentArgument.modifiedDate
                    ),
                })}
            </p>
        </IonCol>
    </IonRow>
</IonGrid>
{currentUser !== null &&
    currentArgument.authorId === currentUser.userId && (
        <IonButton
            routerLink={'/argument/${currentArgument.argumentId}/edit'}
        >
            Edit This Argument
        </IonButton>
    )}

<IonButton
    routerLink={'/argument/${currentArgument.argumentId}/create/attacking'}
>
    Attack This Argument
</IonButton>

<IonButton
    routerLink={'/argument/${currentArgument.argumentId}/create/supporting'}
>
    Support This Argument
</IonButton>
</IonCardContent>
</>
)}
</IonCard>
</IonCol>
</IonRow>
<IonRow>

```

```

<IonCol>
  <IonCard>
    <IonCardHeader>
      <IonCardSubtitle>Comments</IonCardSubtitle>
    </IonCardHeader>
    <IonCardContent>
      <IonList>
        {argumentComments.map((commentTableEntry, index) => (
          <IonItem key={index}>
            <IonLabel className="ion-text-wrap">
              <h3>
                <IonRouterLink
                  routerLink={'/profile/${commentTableEntry.authorId}'}
                >
                  {commentTableEntry.authorDetails.displayName}
                </IonRouterLink>
              </h3>
              <p>{'Created ${timeDifference({
                previous: new Date(commentTableEntry.createdDate),
              })}'</p>
              <IonText>
                <p>{commentTableEntry.commentBody}</p>
              </IonText>
            </IonLabel>
            {currentUser !== null &&
              commentTableEntry.authorId === currentUser.userId && (
                <IonButton
                  size="small"
                  slot="end"
                  onClick={() => {
                    deleteComment(commentTableEntry.commentId);
                  }}
                >
                  Delete
                </IonButton>
              )})
            </IonItem>
          )));
      </IonList>
      <CommentForm argumentId={argumentId} />
    </IonCardContent>
  </IonCard>

```

```
</IonCol>
</IonRow>
</IonGrid>
</IonContent>
</IonPage>
);
};

export default Argument;
```

### E.2.31 argupedia-app-master/src/pages/CreateArgument.css

```
.argupedia-argument-ion-popover {  
    --max-width: 500px;  
    --width: 100%;  
    -ms-text-justify: right;  
}  
  
.argupedia-argument-ion-select-option {  
    width: 500px;  
}  
  
.argupedia-argument-ion-select {  
    width: 500px;  
}
```

## E.2.32 argupedia-app-master/src/pages/CreateArgument.tsx

```
import { Authenticator } from "@aws-amplify/ui-react";
import {
  IonButton,
  IonCard,
  IonCardContent,
  IonCardHeader,
  IonCardSubtitle,
  IonCardTitle,
  IonCol,
  IonContent,
  IonGrid,
  IonIcon,
  IonInput,
  IonItem,
  IonLabel,
  IonList,
  IonListHeader,
  IonPage,
  IonPopover,
  IonRow,
  IonSelect,
  IonSelectOption,
  IonTextarea,
} from "@ionic/react";
import { Auth } from "aws-amplify";
import { Field, FieldProps, Form, Formik, useFormikContext } from "formik";
import { addCircleOutline } from "ionicons/icons";
import _ from "lodash";
import React, { useEffect, useState } from "react";
import { useHistory, useParams } from "react-router";
import Box from "../components/Box";
import Toolbar from "../components/Toolbar";
import { argumentTypePlaceholders } from "../constants/argumentTypePlaceholders";
import { argumentTypeSchema } from "../constants/argumentTypeSchema";
import { argumentTranslations } from "../helpers/argupedia/argumentTranslations";
import { capitalizeFirstLetter } from "../helpers/argupedia/capitalizeFirstLetter";
import { formatCriticalQuestions } from "../helpers/argupedia/formatCriticalQuestions";
import { pascalCaseToSentence } from "../helpers/argupedia/pascalCaseToSentence";
import { timeDifference } from "../helpers/argupedia/timeDifference";
import useStore from "../hooks";
```

```

import api from "../services";
import {
  Argument,
  ArgumentTableEntry,
  argumentTypes,
} from "../types/arguments";
import "./CreateArgument.css";

const CreateArgument: React.FC = () => {
  const storeUser = useStore((state) => state.storeUser);
  const currentUser = useStore((state) => state.currentUser);
  const history = useHistory();

  const initialValues = {
    argumentType: "",
    argumentText: "",
  };
  const [parentArgument, setParentArgument] = useState<ArgumentTableEntry>();

  const { argumentId, attackingOrSupporting } =
    useParams<{ argumentId: string; attackingOrSupporting: string }>();

  useEffect(() => {
    async function getUserDetails() {
      const cognitoSession = await Auth.currentSession();
      const userDetails = await api.getUserById(
        cognitoSession.getIdToken().payload.sub
      );
      storeUser(userDetails);
      return userDetails;
    }

    async function createUser() {
      storeUser(await api.createUser());
    }

    getUserDetails().catch(() => createUser());
  });

  const getParentArgument = async () => {
    const argument = await api.getArgumentWithId(argumentId);
    setParentArgument(argument);
  };
}

```

```

if (argumentId !== undefined) {
  getParentArgument();
}

}, [argumentId, storeUser]);

const IonInputWrapper = ({

  field,
  form,
  ...props
}: FieldProps & { label: string }) => (
  <IonItem>
    <IonLabel>
      <b>{props.label}</b>
    </IonLabel>
    <IonInput
      onIonChange={(e) => form.setFieldValue(field.name, e.detail.value)}
      id={`${field.name}-input`}
      {...props}
      {...field}
    ></IonInput>
    {parentArgument && (
      <IonButton slot="end" color="light" id={`${field.name}-button`}>
        <IonIcon icon={addCircleOutline} slot="icon-only"></IonIcon>
      </IonButton>
    )}
    {parentArgument && (
      <IonPopover trigger={`${field.name}-button`} dismissOnSelect={true}>
        <IonContent>
          <IonList>
            {Object.keys(
              (argumentTypeSchema as any)[parentArgument.argumentType]
            ).map((key, index) => (
              <IonItem
                key={index}
                button={true}
                detail={false}
                onClick={() => {
                  form.setFieldValue(
                    field.name,
                    (parentArgument.argument as any)[key]
                  );
                }}
              >
            ))}
          </IonList>
        </IonContent>
      </IonPopover>
    )}
  )
)

```

```

        })}
      >
        <IonLabel>{(parentArgument.argument as any)[key]}</IonLabel>
      </IonItem>
    ))}
  </IonList>
</IonContent>
</IonPopover>
)
</IonItem>
);

```

```

const UpdatingIonInputWrapper = ({
  field,
  form,
  ...props
}: FieldProps & { label: string; name: string; test: string }) => {
  const { values, setFieldValue } = useFormikContext();

  React.useEffect(() => {
    // set the value of textC, based on textA and textB
    if ((values as any).argumentType !== "") {
      const keys = Object.keys(
        (argumentTypeSchema as any)[(values as any).argumentType]
      );
      const argument = _.pick(values, keys);
      for (const key of keys) {
        if ((argument as any)[key] === undefined)
          (argument as any)[key] = (argumentTypePlaceholders as any)[
            (values as any).argumentType
          ][key];
      }
      setFieldValue(
        field.name,
        argumentTranslations(
          (values as any).argumentType,
          argument as Argument
        )
      );
    }
  }, [field.name, setFieldValue, values]);
}

return (

```

```

<IonItem>
  <IonLabel>
    <b>{props.label}</b>
  </IonLabel>
  <IonTextarea
    onIonChange={(e) => form.setFieldValue(field.name, e.detail.value)}
    {...props}
    {...field}
  ></IonTextarea>
</IonItem>
);

};

const IonSelectWrapper = ({

  field,
  form,
  ...props

}: FieldProps & { label: string }) => (
<IonItem>
  <IonLabel>
    <b>{props.label}</b>
  </IonLabel>
  <IonSelect
    onIonChange={(e) => {
      console.log(form.values);
      form.setFieldValue(field.name, e.detail.value);
    }}
    interface="popover"
    interfaceOptions={{
      size: "auto",
      cssClass: "argupedia-argument-ion-popover",
    }}
    {...props}
    {...field}
  >
    {argumentTypes.map((argumentType, index) => (
      <IonSelectOption value={argumentType} key={index}>
        {pascalCaseToSentence(argumentType)}
      </IonSelectOption>
    )));
  </IonSelect>
</IonItem>
);

```

```

);

const BooleanIonSelectWrapper = ({
  field,
  form,
  ...props
}: FieldProps & { label: string }) => (
  <IonItem>
    <IonLabel>
      <b>{props.label}</b>
    </IonLabel>
    <IonSelect
      onIonChange={(e) => form.setFieldValue(field.name, e.detail.value)}
      {...props}
      {...field}
    >
      {[ "true", "false" ].map((argumentType, index) => (
        <IonSelectOption value={argumentType} key={index}>
          {argumentType}
        </IonSelectOption>
      )))
    </IonSelect>
  </IonItem>
);

const ConsequenceIonSelectWrapper = ({
  field,
  form,
  ...props
}: FieldProps & { label: string }) => (
  <IonItem>
    <IonLabel>
      <b>{props.label}</b>
    </IonLabel>
    <IonSelect
      onIonChange={(e) => form.setFieldValue(field.name, e.detail.value)}
      {...props}
      {...field}
    >
      {[ "good", "bad" ].map((argumentType, index) => (
        <IonSelectOption key={index} value={argumentType}>
          {argumentType}
        </IonSelectOption>
      )))
    </IonSelect>
  </IonItem>
);

```

```

        </IonSelectOption>
    ))
)
</IonSelect>
</IonItem>
);

return (
<IonPage>
<Toolbar />
<IonContent fullscreen>
<Authenticator>
{({ signOut, user }) => (
<>
<IonCard>
{parentArgument && (
<>
<IonCardHeader>
<IonCardSubtitle>
{capitalizeFirstLetter(attackingOrSupporting)} Argument
</IonCardSubtitle>
<IonCardTitle>
{argumentTranslations(
    parentArgument.argumentType,
    parentArgument.argument
)}
</IonCardTitle>
</IonCardHeader>
<IonCardContent>
<IonGrid>
<IonRow>
<IonCol>
<p>Argument Type</p>
</IonCol>
<IonCol>
<p>
{pascalCaseToSentence(
    parentArgument.argumentType
)}
</p>
</IonCol>
</IonRow>
{Object.keys(parentArgument.argument).map(

```

```

        (key, index) => (
          <IonRow key={index}>
            <IonCol>
              <p>{key}</p>
            </IonCol>
            <IonCol>
              <p>{(parentArgument.argument as any)[key]}</p>
            </IonCol>
          </IonRow>
        )
      )}
    <IonRow>
      <IonCol>
        <p>Last Modified</p>
      </IonCol>
      <IonCol>
        <p>
          {timeDifference({
            previous: new Date(parentArgument.modifiedDate),
          })}
        </p>
      </IonCol>
    </IonRow>
  </IonGrid>
</IonCardContent>
</>
)
</IonCard>
{parentArgument && (
  <IonCard>
    <IonCardHeader>
      <IonCardSubtitle>Critical Questions</IonCardSubtitle>
    </IonCardHeader>
    <IonCardContent>
      <p>
        <b>
          Use the following critical questions to consider how you
          would like to respond to the argument:
        </b>
      </p>
      <IonGrid>
        {formatCriticalQuestions(

```

```

parentArgument.argumentType,
parentArgument.argument
).map((question, index) => (
<IonRow key={index}>
<IonCol>
<p>{question}</p>
</IonCol>
</IonRow>
))
)
</IonGrid>
</IonCardContent>
</IonCard>
)
)
<IonCard>
<IonCardHeader>
<IonCardSubtitle>Create New Argument</IonCardSubtitle>
</IonCardHeader>
<IonCardContent>
<Formik<typeof initialValues>
  initialValues={initialValues}
  enableReinitialize={true}
  onSubmit={(values) => {
    const argumentParameter = {
      argumentType: values.argumentType,
      argument: _.omit(values, [
        "argumentType",
        "argumentText",
      ]) as Argument,
      argumentText: values.argumentText,
    };
    if (argumentId === undefined) {
      api.createNewArgument(argumentParameter).then((res) => {
        history.push(`/argument/${res.argumentId}`);
        window.location.reload();
      });
    } else if (attackingOrSupporting === "attacking") {
      api
        .createNewAttackingArgument(
          argumentParameter,
          argumentId
        )
    }
  }}

```

```

    .then((res) => {
      history.push('/argument/${res.argumentId}');
      window.location.reload();
    });
  } else if (attackingOrSupporting === "supporting") {
    api
      .createNewSupportingArgument(
        argumentParameter,
        argumentId
      )
    .then((res) => {
      history.push('/argument/${res.argumentId}');
      window.location.reload();
    });
  };
}

>
{{{{ submitForm, touched, values }}} => (
  <Form>
    <Box py={12}>
      <IonList>
        <IonListHeader>
          <IonLabel>Select argument type</IonLabel>
        </IonListHeader>
        <Field
          name="argumentType"
          label="Argument Type"
          type="text"
          component={IonSelectWrapper}
        />
      </IonList>
    </Box>
    <Box py={12}>
      <IonList>
        <IonListHeader>
          <IonLabel>Fill in argument details</IonLabel>
        </IonListHeader>
        {values.argumentType ? (
          Object.keys(
            (argumentTypeSchema as any)[values.argumentType]
          ).map((key, index) => {
            const getWrapperForArgumentPropositionType = (

```

```

        argumentPropositionType: string
    ) => {
    switch (argumentPropositionType) {
        case "string":
            return IonInputWrapper;
        case "boolean":
            return BooleanIonSelectWrapper;
        case "consequenceType":
            return ConsequenceIonSelectWrapper;
        default:
            return IonInputWrapper;
    }
};

const argumentPropositionType = (
    argumentTypeSchema as any
)[values.argumentType][key];

return (
    <Field
        name={key}
        key={index}
        label={pascalCaseToSentence(key)}
        type="text"
        placeholder={
            (
                (argumentTypePlaceholders as any)[
                    values.argumentType
                ] as any
            )[key]
        }
        component={getWrapperForArgumentPropositionType(
            argumentPropositionType
        )}
    />
);
}) : (
    <IonItem>
        <IonLabel>
            Select an argument type first
        </IonLabel>
    </IonItem>
)

```

```

        )}

      </IonList>

    </Box>

    <Box py={12}>

      <IonList>

        <IonListHeader>
          <IonLabel>Review your argument</IonLabel>
        </IonListHeader>

        <Field
          name={"argumentText"}
          label={"Argument\u2014Text"}
          type="text"
          component={UpdatingIonInputWrapper}
        />

        <IonButton onClick={submitForm}>Submit</IonButton>

      </IonList>

    </Box>

  </Form>

}

</Formik>

</IonCardContent>

</IonCard>

</>

)}

</Authenticator>

</IonContent>

</IonPage>

};

};

export default CreateArgument;

```

### E.2.33 argupedia-app-master/src/pages/EditArgument.css

### E.2.34 argupedia-app-master/src/pages/EditArgument.tsx

```
import { Authenticator } from "@aws-amplify/ui-react";
import {
  IonButton,
  IonContent,
  IonItem,
  IonLabel,
  IonList,
  IonPage,
  IonTextarea,
} from "@ionic/react";

import { Field, FieldProps, Form, Formik } from "formik";
import React, { useEffect, useState } from "react";
import { useHistory, useParams } from "react-router";
import Toolbar from "../components/Toolbar";
import { argumentTranslations } from "../helpers/argupedia/argumentTranslations";
import api from "../services";
import "./EditProfile.css";

const EditArgument: React.FC = () => {
  const history = useHistory();

  const { argumentId } = useParams<{ argumentId: string }>();

  const [initialValues, setInitialValues] = useState({
    argumentText: "",
  });

  useEffect(() => {
    async function getExistingArgumentDetails() {
      const argumentTableEntry = await api.getArgumentWithId(argumentId);
      setInitialValues({
        argumentText:
          argumentTableEntry.argumentText ||
          argumentTranslations(
            argumentTableEntry.argumentType,
            argumentTableEntry.argument
          ),
      });
    }
  }, []);

  return (
    <Formik
      initialValues={initialValues}
      onSubmit={async (values) => {
        const updatedArgument = {
          argumentText: values.argumentText,
        };
        await api.updateArgumentWithId(argumentId, updatedArgument);
        history.push("/");
      }}
    >
      <IonPage>
        <IonContent>
          <Toolbar title="Edit Argument" />
          <IonTextarea
            value={initialValues.argumentText}
            onChange={(e) => setInitialValues({ ...initialValues, argumentText: e.target.value })}
            placeholder="Enter argument text"
          />
          <IonList>
            <IonItem>
              <IonLabel>Type:</IonLabel>
              <IonText>{initialValues.argumentType}</IonText>
            </IonItem>
            <IonItem>
              <IonLabel>Value:</IonLabel>
              <IonText>{initialValues.argument}</IonText>
            </IonItem>
          </IonList>
        </IonContent>
      </IonPage>
    </Formik>
  );
}
```

```

getExistingArgumentDetails();
}, []);

const IonTextareaWrapper = ({

  field,
  form,
  ...props
}: FieldProps & { label: string; name: string; test: string }) => (
  <IonItem>
    <IonLabel position="stacked">
      <b>{props.label}</b>
    </IonLabel>
    <IonTextarea
      onIonChange={(e) => form.setFieldValue(field.name, e.detail.value)}
      {...props}
      {...field}
    ></IonTextarea>
  </IonItem>
);

return (
  <IonPage>
    <Toolbar />
    <IonContent fullscreen>
      <Authenticator>
        {({ signOut, user }) => (
          <>
            <h1>Edit Argument</h1>
            <Formik<typeof initialValues>
              initialValues={initialValues}
              enableReinitialize={true}
              onSubmit={({values}) => {
                api
                  .updateArgumentWithId(argumentId, values.argumentText)
                  .then(() => {
                    history.push(`/argument/${argumentId}`);
                    window.location.reload();
                  });
              }}
            >
              {({ submitForm }) => (
                <Form>

```

```
<IonList>
  <Field
    name="argumentText"
    label="Argument Text"
    type="text"
    component={IonTextareaWrapper}>
  />
  <IonButton onClick={submitForm}>Save</IonButton>
</IonList>
</Form>
)}
</Formik>
</>
)
}

</Authenticator>
</IonContent>
</IonPage>
);

};

export default EditArgument;
```

### E.2.35 argupedia-app-master/src/pages/EditProfile.css

### E.2.36 argupedia-app-master/src/pages/EditProfile.tsx

```
import { Authenticator } from "@aws-amplify/ui-react";
import {
  IonButton,
  IonContent,
  IonInput,
  IonItem,
  IonLabel,
  IonList,
  IonPage,
} from "@ionic/react";
import { Auth } from "aws-amplify";
import { Field, FieldProps, Form, Formik } from "formik";
import React, { useEffect, useState } from "react";
import { useHistory } from "react-router";
import Toolbar from "../components/Toolbar";
import useStore from "../hooks";
import api from "../services";
import { ArgumentTableEntry } from "../types/arguments";
import { CommentTableEntry } from "../types/comments";
import "./EditProfile.css";

const EditProfile: React.FC = () => {
  const storeUser = useStore((state) => state.storeUser);
  const currentUser = useStore((state) => state.currentUser);

  const history = useHistory();

  const [authoredComments, setAuthoredComments] = useState<CommentTableEntry[]>(
    []
  );
  const [authoredArguments, setAuthoredArguments] = useState<
    ArgumentTableEntry[]
  >([]);
  const [votedArguments, setVotedArguments] = useState<ArgumentTableEntry[]>(
    []
  );
  const [initialValues, setInitialValues] = useState({
    displayName: "",
    biography: "",
    profilePictureURL: "",
  
```

```

});
```

```

useEffect(() => {
  async function getUserDetails() {
    const cognitoSession = await Auth.currentSession();
    const userDetails = await api.getUserById(
      cognitoSession.getIdToken().payload.sub
    );
    storeUser(userDetails);
    return userDetails;
  }

  async function createUser() {
    storeUser(await api.createUser());
  }

  getUserDetails().catch(() => createUser());
}, []);
```

```

useEffect(() => {
  // console.log(currentUser);
  if (currentUser !== null)
    setInitialValues({
      displayName: currentUser.displayName, //currentUser.displayName,
      biography: currentUser.biography, //currentUser.biography,
      profilePictureURL: currentUser.profilePictureURL, //currentUser.profilePictureURL,
    });
}, [currentUser]);
```

```

const IonInputWrapper = ({
```

```

  field,
  form,
  ...props
}: FieldProps & { label: string }) => (
  <IonItem>
    <IonLabel>{props.label}</IonLabel>
    <IonInput
      onIonChange={(e) => form.setFieldValue(field.name, e.detail.value)}
      {...props}
      {...field}
    />
  </IonItem>
```

```

);

return (
<IonPage>
<Toolbar />
<IonContent fullscreen>
<Authenticator>
{({ signOut, user }) => (
<>
<h1>Edit Profile</h1>
<Formik<typeof initialValues>
  initialValues={initialValues}
  enableReinitialize={true}
  onSubmit={(values) => {
    if (currentUser !== null && values !== undefined)
      api
        .updateUserWithId(currentUser.userId, {
          displayName: values.displayName,
          biography: values.biography,
        })
        .then(() => {
          history.push("/profile");
          window.location.reload();
        });
  }}
>
{({ submitForm }) => (
<Form>
<IonList>
<Field
  name="displayName"
  label="Display Name"
  type="text"
  component={IonInputWrapper}
/>
<Field
  name="biography"
  label="Biography"
  type="text"
  component={IonInputWrapper}
/>
<IonButton onClick={submitForm}>Save</IonButton>

```

```
        </IonList>
      </Form>
    )}
</Formik>
</>
)
}

</Authenticator>
</IonContent>
</IonPage>
);

};

export default EditProfile;
```

**E.2.37 argupedia-app-master/src/pages/Home.css**

## E.2.38 argupedia-app-master/src/pages/Home.tsx

```
import {
  IonAvatar,
  IonButton,
  IonButtons,
  IonContent,
  IonHeader,
  IonItem,
  IonPage,
  IonText,
  IonTitle,
  IonToolbar,
} from "@ionic/react";

import { Auth } from "aws-amplify";
import React, { useEffect, useState } from "react";
import api from "../services";
import "./Home.css";
import useStore from "../hooks";
import { Authenticator } from "@aws-amplify/ui-react";
import ArgumentCard from "../components/ArgumentCard";
import { argumentTranslations } from "../helpers/argupedia/argumentTranslations";
import { ArgumentTableEntry } from "../types/arguments";
import Toolbar from "../components/Toolbar";

const Home: React.FC = () => {
  const storeUser = useStore((state) => state.storeUser);
  const currentUser = useStore((state) => state.currentUser);
  const [args, setArgs] = useState<ArgumentTableEntry[]>([]);
  useEffect(() => {
    async function getUserDetails() {
      const cognitoSession = await Auth.currentSession();
      const userDetails = await api.getUserById(
        cognitoSession.getIdToken().payload.sub
      );
      storeUser(userDetails);
      return userDetails;
    }
    async function createUser() {
      storeUser(await api.createUser());
    }
  })
}
```

```

getUserDetails().catch(() => createUser());

async function hello() {
  const res = await api.hello();
  console.log(res);
}

hello();

async function getAllArgumentData() {
  setArgs((await api.getAllArguments()).Items);
}

getAllArgumentData();

}, []);

return (
<IonPage>
  <Toolbar />
  <IonContent fullscreen>
    <IonHeader collapse="condense">
      <IonToolbar>
        <IonTitle size="large">Welcome to Argupedia</IonTitle>
      </IonToolbar>
    </IonHeader>
    {args.map((arg) =>
      ArgumentCard({
        argumentText: argumentTranslations(arg.argumentType, arg.argument),
        argumentId: arg.argumentId,
      })
    )}
  </IonContent>
</IonPage>
);
};

export default Home;

```

### E.2.39 argupedia-app-master/src/pages/Profile.css

## E.2.40 argupedia-app-master/src/pages/Profile.tsx

```
import {
  IonAvatar,
  IonButton,
  IonButtons,
  IonCol,
  IonContent,
  IonGrid,
  IonHeader,
  IonImg,
  IonItem,
  IonList,
  IonListHeader,
  IonPage,
  IonRow,
  IonText,
  IonTitle,
  IonToolbar,
} from "@ionic/react";

import { Auth } from "aws-amplify";
import React, { useEffect, useState } from "react";
import api from "../services";
import "./Profile.css";
import useStore from "../hooks";
import { Authenticator } from "@aws-amplify/ui-react";
import { CommentTableEntry } from "../types/comments";
import { ArgumentTableEntry } from "../types/arguments";
import { argumentTranslations } from "../helpers/argupedia/argumentTranslations";
import { useParams } from "react-router";
import { UserTableEntry } from "../types";
import { timeDifference } from "../helpers/argupedia/timeDifference";
import Toolbar from "../components/Toolbar";

const Home: React.FC = () => {
  const storeUser = useStore((state) => state.storeUser);
  const currentUser = useStore((state) => state.currentUser);
  const [authoredComments, setAuthoredComments] = useState<CommentTableEntry[]>(
    []
  );
  const [authoredArguments, setAuthoredArguments] = useState<
    ArgumentTableEntry[]
  
```

```

> ([]);

const [userDetails, setUserDetails] = useState<UserTableEntry>();

const { userId } = useParams<{ userId: string }>();

useEffect(() => {

  async function getUserDetails() {
    const cognitoSession = await Auth.currentSession();
    const userDetails = await api.getUserId(
      cognitoSession.getIdToken().payload.sub
    );
    storeUser(userDetails);
    return userDetails;
  }

  async function createUser() {
    storeUser(await api.createUser());
  }

  getUserDetails().catch(() => createUser());
}, []);

useEffect(() => {

  async function getAuthoredArgumentsForUserWithId(userId: string) {
    const authoredArguments = await api.getAuthoredArgumentsForUserWithId(
      userId
    );
    setAuthoredArguments(authoredArguments);
  }

  async function getAuthoredCommentsForUserWithId(userId: string) {
    const authoredComments = await api.getAuthoredCommentsForUserWithId(
      userId
    );
    setAuthoredComments(authoredComments);
  }

  async function getUserDetails(userId: string) {
    api
      .getUserById(userId)
      .then((userDetails) => {
        setUserDetails(userDetails);
      })
  }
});

```

```

        })
      .catch((e) => {
        console.log(e);
      });
    }

    if (userId !== undefined) {
      getAuthoredArgumentsForUserWithId(userId);
      getAuthoredCommentsForUserWithId(userId);
      getUserDetails(userId);
    } else if (currentUser !== null) {
      console.log(currentUser);
      getAuthoredArgumentsForUserWithId(currentUser.userId);
      getAuthoredCommentsForUserWithId(currentUser.userId);
      getUserDetails(currentUser.userId);
    }
  }, [currentUser]);

const deleteUserAccount = () => {
  if (currentUser !== null) {
    api.deleteUserWithId(currentUser.userId).then(() => {
      storeUser(null);
      Auth.signOut();
    });
  }
};

return (
  <IonPage>
  <Toolbar />
  <IonContent fullscreen>
    <IonGrid>
      <IonRow>
        <IonCol>
          {userId === undefined ||
          (userDetails !== undefined && userId === userDetails.userId) ? (
            <>
              <IonButton routerLink={`/profile/edit`}>
                Edit Profile
              </IonButton>{" "}
              <IonButton
                onClick={() => {

```

```

        deleteUserAccount();
    }
}

Delete Account
</IonButton>
</>
) : (
"""

)}
<h1>Your Profile</h1>
<IonGrid>
<IonRow>
<IonCol>Name</IonCol>
<IonCol>
  {userDetails === undefined ? "" : userDetails.displayName}
</IonCol>
</IonRow>
<IonRow>
<IonCol>Biography</IonCol>
<IonCol>
  {userDetails === undefined ? "" : userDetails.biography}
</IonCol>
</IonRow>
</IonGrid>
</IonCol>
</IonRow>
</IonGrid>
<IonGrid>
<IonRow>
<IonCol>
<IonList>
<IonListHeader>Authored Arguments</IonListHeader>
{authoredArguments.map((argumentTableEntry, index) => (
  <IonItem
    routerLink={'/argument/${argumentTableEntry.argumentId}'}
    key={index}
  >
    <IonText>
      {argumentTableEntry.argumentText ||
        argumentTranslations(
          argumentTableEntry.argumentType,
          argumentTableEntry.argument
        )
      }
    </IonText>
  
```

```

        )}
      </IonText>
    </IonItem>
  )})
</IonList>
</IonCol>
<IonCol>
  <IonList>
    <IonListHeader>Authored Comments</IonListHeader>
    {authoredComments.map((commentTableEntry, index) => (
      <IonItem
        routerLink={`/argument/${commentTableEntry.commentId}`}
        key={index}
      >
        <IonText>{commentTableEntry.commentBody}</IonText>
      </IonItem>
    )));
  </IonList>
</IonCol>
</IonRow>
</IonGrid>
</IonContent>
</IonPage>
);
};

export default Home;

```

## E.2.41 argupedia-app-master/src/react-app-env.d.ts

```
///<reference types="react-scripts"/>
```

## E.2.42 argupedia-app-master/src/reportWebVitals.ts

```
import { ReportHandler } from "web-vitals";

const reportWebVitals = (onPerfEntry?: ReportHandler) => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import("web-vitals").then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;
```

## E.2.43 argupedia-app-master/src/service-worker.ts

```
//> <reference lib="webworker" />
/* eslint-disable no-restricted-globals */

// This service worker can be customized!
// See https://developers.google.com/web/tools/workbox/modules
// for the list of available Workbox modules, or add any other
// code you'd like.
// You can also remove this file if you'd prefer not to use a
// service worker, and the Workbox build step will be skipped.

import { clientsClaim } from "workbox-core";
import { ExpirationPlugin } from "workbox-expiration";
import { precacheAndRoute, createHandlerBoundToURL } from "workbox-precaching";
import { registerRoute } from "workbox-routing";
import { StaleWhileRevalidate } from "workbox-strategies";

declare const self: ServiceWorkerGlobalScope;

clientsClaim();

// Precache all of the assets generated by your build process.
// Their URLs are injected into the manifest variable below.
// This variable must be present somewhere in your service worker file,
// even if you decide not to use precaching. See https://cra.link/PWA
precacheAndRoute(self.__WB_MANIFEST);

// Set up App Shell-style routing, so that all navigation requests
// are fulfilled with your index.html shell. Learn more at
// https://developers.google.com/web/fundamentals/architecture/app-shell
const fileExtensionRegexp = new RegExp("/[^/?]+\\.[^/]+$/");
registerRoute(
  // Return false to exempt requests from being fulfilled by index.html.
  ({ request, url }: { request: Request; url: URL }) => {
    // If this isn't a navigation, skip.
    if (request.mode !== "navigate") {
      return false;
    }

    // If this is a URL that starts with /_, skip.
    if (url.pathname.startsWith("/_")) {

```

```

    return false;
}

// If this looks like a URL for a resource, because it contains
// a file extension, skip.
if (url.pathname.match(fileExtensionRegexp)) {
    return false;
}

// Return true to signal that we want to use the handler.
return true;
},
createHandlerBoundToURL(process.env.PUBLIC_URL + "/index.html")
);

// An example runtime caching route for requests that aren't handled by the
// precache, in this case same-origin .png requests like those from in public/
registerRoute(
    // Add in any other file extensions or routing criteria as needed.
    ({ url }) =>
        url.origin === self.location.origin && url.pathname.endsWith(".png"),
    // Customize this strategy as needed, e.g., by changing to CacheFirst.
    new StaleWhileRevalidate({
        cacheName: "images",
        plugins: [
            // Ensure that once this runtime cache reaches a maximum size the
            // least-recently used images are removed.
            new ExpirationPlugin({ maxEntries: 50 }),
        ],
    })
);

// This allows the web app to trigger skipWaiting via
// registration.waiting.postMessage({type: 'SKIP_WAITING'})
self.addEventListener("message", (event) => {
    if (event.data && event.data.type === "SKIP_WAITING") {
        self.skipWaiting();
    }
});

// Any other custom service worker logic can go here.

```

## E.2.44 argupedia-app-master/src/serviceWorkerRegistration.ts

```
// This optional code is used to register a service worker.  
// register() is not called by default.  
  
// This lets the app load faster on subsequent visits in production, and gives  
// it offline capabilities. However, it also means that developers (and users)  
// will only see deployed updates on subsequent visits to a page, after all the  
// existing tabs open on the page have been closed, since previously cached  
// resources are updated in the background.  
  
// To learn more about the benefits of this model and instructions on how to  
// opt-in, read https://cra.link/PWA  
  
const isLocalhost = Boolean(  
  window.location.hostname === "localhost" ||  
  // [::1] is the IPv6 localhost address.  
  window.location.hostname === "[::1]" ||  
  // 127.0.0.0/8 are considered localhost for IPv4.  
  window.location.hostname.match(  
    /^127(?:\.(?:25[0-5]|2[0-4][0-9]|0[1-9][0-9]|0)|[0-9]{1,3}){3}$/,
  )  
);  
  
type Config = {  
  onSuccess?: (registration: ServiceWorkerRegistration) => void;  
  onUpdate?: (registration: ServiceWorkerRegistration) => void;  
};  
  
export function register(config?: Config) {  
  if (process.env.NODE_ENV === "production" && "serviceWorker" in navigator) {  
    // The URL constructor is available in all browsers that support SW.  
    const publicUrl = new URL(process.env.PUBLIC_URL, window.location.href);  
    if (publicUrl.origin !== window.location.origin) {  
      // Our service worker won't work if PUBLIC_URL is on a different origin  
      // from what our page is served on. This might happen if a CDN is used to  
      // serve assets; see https://github.com/facebook/create-react-app/issues/2374  
      return;  
    }  
  
    window.addEventListener("load", () => {  
      const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;  
    });  
  }  
}
```

```

if (isLocalhost) {
  // This is running on localhost. Let's check if a service worker still exists or not.
  checkValidServiceWorker(swUrl, config);

  // Add some additional logging to localhost, pointing developers to the
  // service worker/PWA documentation.
  navigator.serviceWorker.ready.then(() => {
    console.log(
      "This web app is being served cache-first by a service worker" +
      "worker. To learn more, visit https://cra.link/PWA"
    );
  });
} else {
  // Is not localhost. Just register service worker
  registerValidSW(swUrl, config);
}
}

function registerValidSW(swUrl: string, config?: Config) {
  navigator.serviceWorker
    .register(swUrl)
    .then((registration) => {
      registration.onupdatefound = () => {
        const installingWorker = registration.installing;
        if (installingWorker === null) {
          return;
        }
        installingWorker.onstatechange = () => {
          if (installingWorker.state === "installed") {
            if (navigator.serviceWorker.controller) {
              // At this point, the updated precached content has been fetched,
              // but the previous service worker will still serve the older
              // content until all client tabs are closed.
              console.log(
                "New content is available and will be used when all" +
                "tabs for this page are closed. See https://cra.link/PWA."
              );
            }
          }
        };
      }
    });
}

```

```

        if (config && config.onUpdate) {
            config.onUpdate(registration);
        }
    } else {
        // At this point, everything has been precached.
        // It's the perfect time to display a
        // "Content is cached for offline use." message.
        console.log("Content is cached for offline use.");

        // Execute callback
        if (config && config.onSuccess) {
            config.onSuccess(registration);
        }
    }
}

};

});

.catch((error) => {
    console.error("Error during service worker registration:", error);
});

}

function checkValidServiceWorker(swUrl: string, config?: Config) {
    // Check if the service worker can be found. If it can't reload the page.
    fetch(swUrl, {
        headers: { "Service-Worker": "script" },
    })
    .then((response) => {
        // Ensure service worker exists, and that we really are getting a JS file.
        const contentType = response.headers.get("content-type");
        if (
            response.status === 404 ||
            (contentType != null && contentType.indexOf("javascript") === -1)
        ) {
            // No service worker found. Probably a different app. Reload the page.
            navigator.serviceWorker.ready.then((registration) => {
                registration.unregister().then(() => {
                    window.location.reload();
                });
            });
        } else {

```

```

// Service worker found. Proceed as normal.
registerValidSW(swUrl, config);
}

})
.catch(() => {
  console.log(
    "No internet connection found. App is running in offline mode."
  );
});

}

export function unregister() {
  if ("serviceWorker" in navigator) {
    navigator.serviceWorker.ready
      .then((registration) => {
        registration.unregister();
      })
      .catch((error) => {
        console.error(error.message);
      });
  }
}

```

## E.2.45 argupedia-app-master/src/services/api/api.ts

```
import { Auth } from "aws-amplify";
import { UserTableResponse } from "../../types";
import {
  Argument,
  ArgumentTableEntry,
  CreateNewArgumentParameters,
  CreateNewArgumentResponse,
  GetAllArgumentsResponse,
  GetArgumentGraphForArgumentWithIdResponse,
  GetArgumentWithIdResponse,
  GetAuthoredArgumentsForUserWithIdResponse,
  GetVotedArgumentsForUserWithIdResponse,
  UpdateArgumentWithIdResponse,
} from "../../types/arguments";
import {
  CreateCommentForArgumentWithIdResponse,
  GetAuthoredCommentsForUserWithIdResponse,
  GetCommentsForArgumentWithIdResponse,
} from "../../types/comments";
import { UpdateUserBody } from "../../types/users";

const {
  REACT_APP_API_BASE_URL:
  baseUrl = "https://6fbzgqfvw0.execute-api.eu-west-2.amazonaws.com/dev",
} = process.env;

const hello = async () => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();
  const response = await fetch(`${baseUrl}/hello`, {
    method: "POST",
    body: JSON.stringify({
      name: "Jonathan",
    }),
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
  });
};
```

```

const createNewArgument = async (argument: CreateNewArgumentParameters) => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();

  const response = await fetch(`${baseUrl}/rest/api/1/argument/`, {
    method: "POST",
    body: JSON.stringify(argument),
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
  });
}

const { message }: CreateNewArgumentResponse = await response.json();

return message;
};

const createNewAttackingArgument = async (
  argument: CreateNewArgumentParameters,
  attackingArgumentId: string
) => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();

  const response = await fetch(
    `${baseUrl}/rest/api/1/argument/${attackingArgumentId}/attacking`,
    {
      method: "POST",
      body: JSON.stringify(argument),
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    },
  );
}

const { message }: CreateNewArgumentResponse = await response.json();

return message;
};

const createNewSupportingArgument = async (
  argument: CreateNewArgumentParameters,

```

```

supportingArgumentId: string
) => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();

  const response = await fetch(
    `${baseUrl}/rest/api/1/argument/${supportingArgumentId}/supporting`,
    {
      method: "POST",
      body: JSON.stringify(argument),
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    },
  );
}

const { message }: CreateNewArgumentResponse = await response.json();

return message;
};

const createUser = async () => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();

  const response = await fetch(`${baseUrl}/rest/api/1/user/`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify({}),
  });

  const { message }: UserTableResponse = await response.json();

  return message;
};

const generateFakeData = async () => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();

  const response = await fetch(`${baseUrl}/rest/api/1/generateFakeData/`, {

```

```

method: "POST",
headers: {
  "Content-Type": "application/json",
  Authorization: `Bearer ${token}`,
},
body: JSON.stringify({}),
});

};

const getUserById = async (userId: string) => {
  const response = await fetch(`${baseUrl}/rest/api/1/user/${userId}`, {
    method: "GET",
  }).catch(() => {
    throw new Error("User not found");
  });

  const { message }: UserTableResponse = await response.json();

  return message;
};

const getAllArguments = async () => {
  const response = await fetch(`${baseUrl}/rest/api/1/argument`, {
    method: "GET",
  });

  const { message }: GetAllArgumentsResponse = await response.json();

  return message;
};

const getAuthoredArgumentsForUserWithId = async (userId: string) => {
  const response = await fetch(
    `${baseUrl}/rest/api/1/user/${userId}/authoredArguments?count=10`,
    {
      method: "GET",
    }
  );

  const { message }: GetAuthoredArgumentsForUserWithIdResponse =
    await response.json();

```

```

    return message;
};

const getAuthoredCommentsForUserWithId = async (userId: string) => {
    const response = await fetch(
        `${baseUrl}/rest/api/1/user/${userId}/authoredComments?count=10`,
        {
            method: "GET",
        }
    );

    const { message }: GetAuthoredCommentsForUserWithIdResponse =
        await response.json();

    return message;
};

const getVotedArgumentsForUserWithId = async (userId: string) => {
    const response = await fetch(
        `${baseUrl}/rest/api/1/user/${userId}/votedArguments?count=10`,
        {
            method: "GET",
        }
    );

    const { message }: GetVotedArgumentsForUserWithIdResponse =
        await response.json();

    return message;
};

const updateUserWithId = async (
    userId: string,
    updateUserBody: UpdateUserBody
) => {
    const token = (await Auth.currentSession()).getIdToken().getJwtToken();

    const response = await fetch(`${baseUrl}/rest/api/1/user/${userId}`, {
        method: "PATCH",
        headers: {
            "Content-Type": "application/json",
            Authorization: `Bearer ${token}`,
        }
    });
}

```

```

    },
    body: JSON.stringify(updateUserBody),
  });

const { message }: UserTableResponse = await response.json();

return message;
};

const getArgumentGraphForArgumentWithId = async (argumentId: string) => {
  const response = await fetch(
    `${baseUrl}/rest/api/1/argument/${argumentId}/graph`,
    {
      method: "GET",
    }
  );
}

const { message }: GetArgumentGraphForArgumentWithIdResponse =
  await response.json();

return message;
};

const getArgumentWithId = async (argumentId: string) => {
  const response = await fetch(`${baseUrl}/rest/api/1/argument/${argumentId}`, {
    method: "GET",
  });
}

const { message }: GetArgumentWithIdResponse = await response.json();

return message;
};

const getCommentsForArgumentWithId = async (argumentId: string) => {
  const response = await fetch(
    `${baseUrl}/rest/api/1/argument/${argumentId}/comments`,
    {
      method: "GET",
    }
  );
}

const { message }: GetCommentsForArgumentWithIdResponse =

```

```

    await response.json();

    return message;
};

const createCommentForArgumentWithId = async (
  argumentId: string,
  commentBody: string
) => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();

  const response = await fetch(
    `${baseUrl}/rest/api/1/argument/${argumentId}/comments`,
    {
      method: "POST",
      body: JSON.stringify({
        commentBody: commentBody,
      }),
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    }
  );
};

const { message }: CreateCommentForArgumentWithIdResponse =
  await response.json();

return message;
};

const updateArgumentWithId = async (
  argumentId: string,
  argumentText: string
) => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();

  const response = await fetch(`${baseUrl}/rest/api/1/argument/${argumentId}`, {
    method: "PATCH",
    body: JSON.stringify({
      argumentText: argumentText,
    }),
  });
};

```

```

headers: {
  "Content-Type": "application/json",
  Authorization: 'Bearer ${token}',
},
});

const { message }: UpdateArgumentWithIdResponse = await response.json();

return message;
};

const deleteCommentWithId = async (commentId: string) => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();

  await fetch(`${baseUrl}/rest/api/1/comment/${commentId}`, {
    method: "DELETE",
    headers: {
      Authorization: 'Bearer ${token}',
    },
  });
};

const deleteUserWithId = async (userId: string) => {
  const token = (await Auth.currentSession()).getIdToken().getJwtToken();

  await fetch(`${baseUrl}/rest/api/1/user/${userId}`, {
    method: "DELETE",
    headers: {
      Authorization: 'Bearer ${token}',
    },
  });
};

const api = {
  createNewArgument,
  hello,
  getUserById,
  createUser,
  generateFakeData,
  getAllArguments,
  getAuthoredArgumentsForUserWithId,
  getAuthoredCommentsForUserWithId,
}

```

```
getVotedArgumentsForUserWithId,  
updateUserWithId,  
getArgumentGraphForArgumentWithId,  
getArgumentWithId,  
createNewAttackingArgument,  
createNewSupportingArgument,  
getCommentsForArgumentWithId,  
createCommentForArgumentWithId,  
updateArgumentWithId,  
deleteCommentWithId,  
deleteUserWithId,  
};  
  
export default api;
```

## E.2.46 argupedia-app-master/src/services/api/index.ts

```
export { default } from "./api";
```

## E.2.47 argupedia-app-master/src/services/index.ts

```
export { default } from "./api";
```

## E.2.48 argupedia-app-master/src/setupTests.ts

```
// jest-dom adds custom jest matchers for asserting on DOM nodes.  
// allows you to do things like:  
// expect(element).toHaveTextContent(/react/i)  
// learn more: https://github.com/testing-library/jest-dom  
import "@testing-library/jest-dom/extend-expect";  
  
// Mock matchmedia  
window.matchMedia =  
  window.matchMedia ||  
  function () {  
    return {  
      matches: false,  
      addListener: function () {},  
      removeListener: function () {},  
    };  
};  
};
```

## E.2.49 argupedia-app-master/src/theme/variables.css

```
/* Ionic Variables and Theming. For more info, please see:  
http://ionicframework.com/docs/theming/ */
```

```
/** Ionic CSS Variables **/  
  
:root {  
  
    /** primary **/  
  
    --ion-color-primary: #3880ff;  
    --ion-color-primary-rgb: 56, 128, 255;  
    --ion-color-primary-contrast: #ffffff;  
    --ion-color-primary-contrast-rgb: 255, 255, 255;  
    --ion-color-primary-shade: #3171e0;  
    --ion-color-primary-tint: #4c8dff;  
  
    /** secondary **/  
    --ion-color-secondary: #3dc2ff;  
    --ion-color-secondary-rgb: 61, 194, 255;  
    --ion-color-secondary-contrast: #ffffff;  
    --ion-color-secondary-contrast-rgb: 255, 255, 255;  
    --ion-color-secondary-shade: #36abe0;  
    --ion-color-secondary-tint: #50c8ff;  
  
    /** tertiary **/  
    --ion-color-tertiary: #5260ff;  
    --ion-color-tertiary-rgb: 82, 96, 255;  
    --ion-color-tertiary-contrast: #ffffff;  
    --ion-color-tertiary-contrast-rgb: 255, 255, 255;  
    --ion-color-tertiary-shade: #4854e0;  
    --ion-color-tertiary-tint: #6370ff;  
  
    /** success **/  
    --ion-color-success: #2dd36f;  
    --ion-color-success-rgb: 45, 211, 111;  
    --ion-color-success-contrast: #ffffff;  
    --ion-color-success-contrast-rgb: 255, 255, 255;  
    --ion-color-success-shade: #28ba62;  
    --ion-color-success-tint: #42d77d;  
  
    /** warning **/  
    --ion-color-warning: #ffc409;  
    --ion-color-warning-rgb: 255, 196, 9;
```

```

--ion-color-warning-contrast: #000000;
--ion-color-warning-contrast-rgb: 0, 0, 0;
--ion-color-warning-shade: #e0ac08;
--ion-color-warning-tint: #ffca22;

/** danger */
--ion-color-danger: #eb445a;
--ion-color-danger-rgb: 235, 68, 90;
--ion-color-danger-contrast: #ffffff;
--ion-color-danger-contrast-rgb: 255, 255, 255;
--ion-color-danger-shade: #cf3c4f;
--ion-color-danger-tint: #ed576b;

/** dark */
--ion-color-dark: #222428;
--ion-color-dark-rgb: 34, 36, 40;
--ion-color-dark-contrast: #ffffff;
--ion-color-dark-contrast-rgb: 255, 255, 255;
--ion-color-dark-shade: #1e2023;
--ion-color-dark-tint: #383a3e;

/** medium */
--ion-color-medium: #92949c;
--ion-color-medium-rgb: 146, 148, 156;
--ion-color-medium-contrast: #ffffff;
--ion-color-medium-contrast-rgb: 255, 255, 255;
--ion-color-medium-shade: #808289;
--ion-color-medium-tint: #9d9fa6;

/** light */
--ion-color-light: #f4f5f8;
--ion-color-light-rgb: 244, 245, 248;
--ion-color-light-contrast: #000000;
--ion-color-light-contrast-rgb: 0, 0, 0;
--ion-color-light-shade: #d7d8da;
--ion-color-light-tint: #f5f6f9;
}

@media (prefers-color-scheme: dark) {
/*
 * Dark Colors
 * -----

```

```
*/  
  
body {  
  
--ion-color-primary: #428cff;  
--ion-color-primary-rgb: 66,140,255;  
--ion-color-primary-contrast: #ffffff;  
--ion-color-primary-contrast-rgb: 255,255,255;  
--ion-color-primary-shade: #3a7be0;  
--ion-color-primary-tint: #5598ff;  
  
--ion-color-secondary: #50c8ff;  
--ion-color-secondary-rgb: 80,200,255;  
--ion-color-secondary-contrast: #ffffff;  
--ion-color-secondary-contrast-rgb: 255,255,255;  
--ion-color-secondary-shade: #46b0e0;  
--ion-color-secondary-tint: #62ceff;  
  
--ion-color-tertiary: #6a64ff;  
--ion-color-tertiary-rgb: 106,100,255;  
--ion-color-tertiary-contrast: #ffffff;  
--ion-color-tertiary-contrast-rgb: 255,255,255;  
--ion-color-tertiary-shade: #5d58e0;  
--ion-color-tertiary-tint: #7974ff;  
  
--ion-color-success: #2fdf75;  
--ion-color-success-rgb: 47,223,117;  
--ion-color-success-contrast: #000000;  
--ion-color-success-contrast-rgb: 0,0,0;  
--ion-color-success-shade: #29c467;  
--ion-color-success-tint: #44e283;  
  
--ion-color-warning: #ffd534;  
--ion-color-warning-rgb: 255,213,52;  
--ion-color-warning-contrast: #000000;  
--ion-color-warning-contrast-rgb: 0,0,0;  
--ion-color-warning-shade: #e0bb2e;  
--ion-color-warning-tint: #ffd948;  
  
--ion-color-danger: #ff4961;  
--ion-color-danger-rgb: 255,73,97;  
--ion-color-danger-contrast: #ffffff;  
--ion-color-danger-contrast-rgb: 255,255,255;
```

```

--ion-color-danger-shade: #e04055;
--ion-color-danger-tint: #ff5b71;

--ion-color-dark: #f4f5f8;
--ion-color-dark-rgb: 244,245,248;
--ion-color-dark-contrast: #000000;
--ion-color-dark-contrast-rgb: 0,0,0;
--ion-color-dark-shade: #d7d8da;
--ion-color-dark-tint: #f5f6f9;

--ion-color-medium: #989aa2;
--ion-color-medium-rgb: 152,154,162;
--ion-color-medium-contrast: #000000;
--ion-color-medium-contrast-rgb: 0,0,0;
--ion-color-medium-shade: #86888f;
--ion-color-medium-tint: #a2a4ab;

--ion-color-light: #222428;
--ion-color-light-rgb: 34,36,40;
--ion-color-light-contrast: #ffffff;
--ion-color-light-contrast-rgb: 255,255,255;
--ion-color-light-shade: #1e2023;
--ion-color-light-tint: #383a3e;
}

/*
 * iOS Dark Theme
 * -----
 */

```

```

.ios body {
  --ion-background-color: #000000;
  --ion-background-color-rgb: 0,0,0;

  --ion-text-color: #ffffff;
  --ion-text-color-rgb: 255,255,255;

  --ion-color-step-50: #0d0d0d;
  --ion-color-step-100: #1a1a1a;
  --ion-color-step-150: #262626;
  --ion-color-step-200: #333333;
  --ion-color-step-250: #404040;
}
```

```

--ion-color-step-300: #4d4d4d;
--ion-color-step-350: #595959;
--ion-color-step-400: #666666;
--ion-color-step-450: #737373;
--ion-color-step-500: #808080;
--ion-color-step-550: #8c8c8c;
--ion-color-step-600: #999999;
--ion-color-step-650: #a6a6a6;
--ion-color-step-700: #b3b3b3;
--ion-color-step-750: #bfbfbf;
--ion-color-step-800: #cccccc;
--ion-color-step-850: #d9d9d9;
--ion-color-step-900: #e6e6e6;
--ion-color-step-950: #f2f2f2;

--ion-item-background: #000000;

--ion-card-background: #1c1c1c;
}

.ios ion-modal {
  --ion-background-color: var(--ion-color-step-100);
  --ion-toolbar-background: var(--ion-color-step-150);
  --ion-toolbar-border-color: var(--ion-color-step-250);
}

/*
 * Material Design Dark Theme
 * -----
 */
.md body {
  --ion-background-color: #121212;
  --ion-background-color-rgb: 18,18,18;

  --ion-text-color: #ffffff;
  --ion-text-color-rgb: 255,255,255;

  --ion-border-color: #222222;

  --ion-color-step-50: #1e1e1e;
}

```

```
--ion-color-step-100: #2a2a2a;  
--ion-color-step-150: #363636;  
--ion-color-step-200: #414141;  
--ion-color-step-250: #4d4d4d;  
--ion-color-step-300: #595959;  
--ion-color-step-350: #656565;  
--ion-color-step-400: #717171;  
--ion-color-step-450: #7d7d7d;  
--ion-color-step-500: #898989;  
--ion-color-step-550: #949494;  
--ion-color-step-600: #a0a0a0;  
--ion-color-step-650: #acacac;  
--ion-color-step-700: #b8b8b8;  
--ion-color-step-750: #c4c4c4;  
--ion-color-step-800: #d0d0d0;  
--ion-color-step-850: #dbdbdb;  
--ion-color-step-900: #e7e7e7;  
--ion-color-step-950: #f3f3f3;  
  
--ion-item-background: #1e1e1e;  
  
--ion-toolbar-background: #1f1f1f;  
  
--ion-tab-bar-background: #1f1f1f;  
  
--ion-card-background: #1e1e1e;  
}  
}
```

## E.2.50 argupedia-app-master/src/types/arguments.ts

```
type ActionArgument = {
    circumstance: string;
    action: string;
    newCircumstance: string;
    goal: string;
    value: string;
};

type ArgumentFromPositionToKnow = {
    person: string;
    fact: string;
    proposition: boolean;
};

type AppealToExpertOpinion = {
    expert: string;
    domain: string;
    fact: string;
    proposition: boolean;
};

type AppealToPopularOpinion = {
    fact: string;
    proposition: boolean;
};

type ArgumentFromAnalogy = {
    originalCase: string;
    similarCase: string;
    fact: string;
    proposition: boolean;
};

type ArgumentFromCorrelationToCause = {
    cause: string;
    effect: string;
};

type ArgumentFromPositiveOrNegativeConsequences = {
    cause: string;
}
```

```

effect: string;
consequence: "GOOD" | "BAD";
};

type SlipperySlopeArgument = {
  firstStepPremise: string;
  recursivePremise: string;
  consequence: "GOOD" | "BAD";
};

type ArgumentFromSign = {
  finding: string;
  fact: string;
  proposition: boolean;
};

type ArgumentFromCommitment = {
  entity: string;
  existingCommitment: string;
  newCommitment: string;
};

type ArgumentFromInconsistentCommitment = {
  entity: string;
  commitment: string;
  otherEvidence: string;
};

type DirectAdHominemArgument = {
  entity: string;
  criticism: string;
};

type CircumstantialAdHominemArgument = {
  entity: string;
  originalArgument: string;
  otherEvidence: string;
};

type ArgumentFromVerbalClassification = {
  entity: string;
  property: string;
};

```

```

classification: string;
};

type Argument =
| ActionArgument
| ArgumentFromPositionToKnow
| AppealToExpertOpinion
| AppealToPopularOpinion
| ArgumentFromAnalogy
| ArgumentFromCorrelationToCause
| ArgumentFromPositiveOrNegativeConsequences
| SlipperySlopeArgument
| ArgumentFromSign
| ArgumentFromCommitment
| ArgumentFromInconsistentCommitment
| DirectAdHominemArgument
| CircumstantialAdHominemArgument
| ArgumentFromVerbalClassification;

const argumentTypes = [
  "ActionArgument",
  "ArgumentFromPositionToKnow",
  "AppealToExpertOpinion",
  "AppealToPopularOpinion",
  "ArgumentFromAnalogy",
  "ArgumentFromCorrelationToCause",
  "ArgumentFromPositiveOrNegativeConsequences",
  "SlipperySlopeArgument",
  "ArgumentFromSign",
  "ArgumentFromCommitment",
  "ArgumentFromInconsistentCommitment",
  "DirectAdHominemArgument",
  "CircumstantialAdHominemArgument",
  "ArgumentFromVerbalClassification",
];

type ArgumentTableEntry = {
  argumentId: string;
  argumentType: string;
  argument: Argument;
  argumentText?: string;
  authorId: string;
}

```

```

createdDate: string;
modifiedDate: string;
supportsArgumentId?: string;
supportedByArgumentIds: Array<string>;
attacksArgumentId?: string;
attackedByArgumentIds: Array<string>;
upvotedByUserIds: Array<string>;
downvotedByUserIds: Array<string>;
label: "IN" | "OUT" | "UNDECIDED";
commentIds: Array<string>;
argumentDepth: number;
};

type CreateNewArgumentHandlerBody = {
  argumentType: string;
  argument: Argument;
};

type GetAllArgumentsResponse = {
  message: {
    Count: number;
    Items: Array<ArgumentTableEntry>;
    ScannedCount: number;
  };
};

type GetAuthoredArgumentsForUserWithIdResponse = {
  message: Array<ArgumentTableEntry>;
};

type GetVotedArgumentsForUserWithIdResponse = {
  message: Array<ArgumentTableEntry>;
};

type GetArgumentGraphForArgumentWithIdResponse = {
  message: Array<ArgumentTableEntry>;
};

type GetArgumentWithIdResponse = {
  message: ArgumentTableEntry;
};

type CreateNewArgumentParameters = {
  argumentType: string;
  argument: Argument;
};

```

```

argumentText: string;
};

type CreateNewArgumentResponse = {
  message: ArgumentTableEntry;
};

type UpdateArgumentWithIdResponse = {
  message: ArgumentTableEntry;
};

export { argumentTypes };
export type {
  ActionArgument,
  AppealToExpertOpinion,
  AppealToPopularOpinion,
  Argument,
  ArgumentFromAnalogy,
  ArgumentFromCommitment,
  ArgumentFromCorrelationToCause,
  ArgumentFromInconsistentCommitment,
  ArgumentFromPositionToKnow,
  ArgumentFromPositiveOrNegativeConsequences,
  ArgumentFromSign,
  ArgumentFromVerbalClassification,
  ArgumentTableEntry,
  CircumstantialAdHominemArgument,
  CreateNewArgumentHandlerBody,
  DirectAdHominemArgument,
  SlipperySlopeArgument,
  GetAllArgumentsResponse,
  GetAuthoredArgumentsForUserWithIdResponse,
  GetVotedArgumentsForUserWithIdResponse,
  GetArgumentGraphForArgumentWithIdResponse,
  GetArgumentWithIdResponse,
  CreateNewArgumentParameters,
  CreateNewArgumentResponse,
  UpdateArgumentWithIdResponse,
};

```

## E.2.51 argupedia-app-master/src/types/comments.ts

```
import { UserTableEntry } from ".";

type CommentTableEntry = {
    commentId: string;
    commentBody: string;
    createdDate: string;
    modifiedDate: string;
    authorId: string;
    argumentId: string;
};

type ArgumentComments = {
    commentId: string;
    commentBody: string;
    createdDate: string;
    modifiedDate: string;
    authorId: string;
    argumentId: string;
    authorDetails: UserTableEntry;
};

type GetAuthoredCommentsForUserWithIdResponse = {
    message: Array<CommentTableEntry>;
};

type GetCommentsForArgumentWithIdResponse = {
    message: Array<ArgumentComments>;
};

type CreateCommentForArgumentWithIdResponse = {
    message: CommentTableEntry;
};

export type {
    CommentTableEntry,
    ArgumentComments,
    GetAuthoredCommentsForUserWithIdResponse,
    GetCommentsForArgumentWithIdResponse,
    CreateCommentForArgumentWithIdResponse,
};
```

## E.2.52 argupedia-app-master/src/types/index.ts

```
export type UserTableEntry = {
    userId: string;
    authoredArgumentIds: Array<string>;
    authoredCommentIds: Array<string>;
    votedArgumentIds: Array<string>;
    displayName: string;
    biography: string;
    profilePictureURL: string;
    lastSignedOn: string;
};

export type UserTableResponse = {
    message: UserTableEntry;
};
```

### E.2.53 argupedia-app-master/src/types/recursivePartial.ts

```
export type RecursivePartial<T> = {
  [P in keyof T]?: T[P] extends (infer U)[] ?
    RecursivePartial<U>[]
  : T[P] extends object
  ? RecursivePartial<T[P]>
  : T[P];
};
```

## E.2.54 argupedia-app-master/src/types/users.ts

```
export type UserTableEntry = {
    userId: string;
    authoredArgumentIds: Array<string>;
    authoredCommentIds: Array<string>;
    votedArgumentIds: Array<string>;
    displayName: string;
    biography: string;
    profilePictureURL: string;
    lastSignedOn: string;
};

export type UpdateUserBody = {
    biography?: string;
    displayName?: string;
    profilePictureURL?: string;
};

export type UserTableResponse = {
    message: UserTableEntry;
};
```

## E.2.55 argupedia-app-master/tsconfig.json

```
{  
  "compilerOptions": {  
    "target": "es5",  
    "lib": [  
      "dom",  
      "dom.iterable",  
      "esnext"  
    ],  
    "allowJs": true,  
    "skipLibCheck": true,  
    "esModuleInterop": true,  
    "allowSyntheticDefaultImports": true,  
    "strict": true,  
    "forceConsistentCasingInFileNames": true,  
    "noFallthroughCasesInSwitch": true,  
    "module": "esnext",  
    "moduleResolution": "node",  
    "resolveJsonModule": true,  
    "isolatedModules": true,  
    "noEmit": true,  
    "jsx": "react-jsx"  
  },  
  "include": [  
    "src"  
  ]  
}
```