

# Examensarbete

Dataingenjör 180hp



## Azure Policy Definition Builder

### Examensarbete 15hp

Halmstad 2021-06-06

Kevin Brandhild



## Preface

I would like to thank my supervisor Galina Sidorenko for helping me through the project. With her guidance, the thesis progressed smoothly. I would also like to thank Ombori for allowing me to tackle this project. Specially Zsolt Háló who helped me when I got stuck and listened to the different ideas and thoughts regarding the project.



## Abstract

Cloud technologies are spearheading today's innovation and automation efforts. With the use of Azure Policy it is possible to govern Azure resources and services through customized Azure policy definitions.

This project aims to simplify the creation process for the Azure policy definition by creating a web-application that removes the need to construct the JSON structure. Instead of code-based, it uses click&add to create the code blocks.

The web-application was created through the library React and with the use of Typescript as the main programming language.

The result is a web-application that removes the need to construct the JSON structure and instead produces this JSON structure for the user.

The project concludes that the result web-application did not fully achieve simplification through click&add. However, it creates a good basis for further development of policy simplification. In its current form, the app can be viewed as an alternative method for creating an Azure policy definition.



## Sammanfattning

Molnteknologierna står i spetsen för dagens innovations- och automatiseringsinsatser. Med användning av Azure Policy är det möjligt att styra Azure-resurser och tjänster genom anpassade Azure-policydefinitioner.

Detta projekt syftar till att förenkla skapandeprocessen för Azure-policydefinitionen genom att skapa en webbapplikation som tar bort behovet av att konstruera JSON struktur. Istället för att strukturera JSON via kod använder den click&add för att skapa kodblocken.

Webbapplikationen skapades genom biblioteket React med programmeringsspråket TypeScript.

Resultatet är en webbapplikation som tar bort behovet av att konstruera JSON strukturen och istället producerar den här JSON-strukturen för användaren.

Projektet drar slutsatsen att resultatapplikationen inte helt nådde förenkling genom användning av klicka&lägg till. Det skapar dock en bra grund för vidareutveckling av förenkling för Azure-principdefinition. I sin nuvarande form kan appen ses som en alternativ metod för att skapa en Azure-principdefinition.





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Customer . . . . .	1
1.2	Purpose and Goal . . . . .	2
1.3	Limitations . . . . .	2
<b>2</b>	<b>Background</b>	<b>3</b>
2.1	Similar existing techniques . . . . .	3
2.1.1	Game Maker Drag&Drop . . . . .	3
2.1.2	Scratch . . . . .	3
2.2	Azure . . . . .	3
2.3	Azure Governance . . . . .	4
2.3.1	Tenant . . . . .	4
2.3.2	Subscription . . . . .	4
2.3.3	Management Group . . . . .	4
2.3.4	Resources . . . . .	4
2.3.5	Resource Group . . . . .	4
2.3.6	Azure Policy . . . . .	4
2.3.7	Azure Policy Definition Structure . . . . .	6
2.3.8	Initiatives . . . . .	7
2.4	Software . . . . .	8
2.4.1	TypeScript . . . . .	8
2.4.2	Cascading Style Sheets(CSS) . . . . .	8
2.4.3	HyperText Markup Language(HTML) . . . . .	8
2.5	Platforms & Libraries . . . . .	8
2.5.1	React . . . . .	8
2.5.2	React-dnd-Beautiful . . . . .	8
2.5.3	React-hook-form . . . . .	10
2.6	Framework . . . . .	11
2.6.1	TreeView . . . . .	11
2.6.2	Tabs . . . . .	11
2.6.3	Other Components . . . . .	11
<b>3</b>	<b>Methods</b>	<b>13</b>
3.1	Pilot Study . . . . .	13
3.1.1	Testing the project . . . . .	14
3.2	Development of the System . . . . .	15
3.2.1	Azure Definition Structure for the application . . . . .	15
3.2.2	Creating the application and components . . . . .	15
3.2.3	Parameter functionality . . . . .	16
3.2.4	PolicyRule functionality . . . . .	18
3.2.5	Drag&Drop or Click&Add . . . . .	20
3.2.6	Menu functionality . . . . .	21
3.2.7	The resulting JSON data . . . . .	21
3.2.8	Design . . . . .	22

3.2.9	Feedback changes . . . . .	22
<b>4</b>	<b>Results</b>	<b>23</b>
4.1	Website & Website Sections . . . . .	23
4.1.1	Parameters . . . . .	24
4.1.2	Menu . . . . .	26
4.1.3	Policy Rule . . . . .	27
4.1.4	Mode & Get JSON . . . . .	29
4.2	Test Results . . . . .	31
<b>5</b>	<b>Discussion</b>	<b>33</b>
5.1	Result compared to goals . . . . .	33
5.2	Flaws in the project . . . . .	34
5.3	Strengths in the project . . . . .	34
5.4	Assessment and evaluation of the project . . . . .	34
5.4.1	Implementation . . . . .	34
5.4.2	Technical Solution . . . . .	35
5.5	Social requirements for technical product development . . . . .	35
5.5.1	Economical . . . . .	35
5.5.2	Environmental . . . . .	35
5.5.3	Security . . . . .	36
<b>6</b>	<b>Conclusion</b>	<b>37</b>
6.1	Improvements . . . . .	37
6.2	Validation from Ombori . . . . .	38
	<b>References</b>	<b>39</b>
<b>7</b>	<b>Appendices</b>	<b>43</b>
7.1	Appendix A . . . . .	43
7.2	Appendix B . . . . .	45

# 1 Introduction

Cloud technologies are spearheading today's innovation and automation efforts. With great speed and power comes great responsibility. Cloud governance is about making sure the resources and services are used in adherence to compliance and security requirements. There are tools to measure, review, or enforce these things. One of these tools exists in the cloud platform Azure [1], where you can use the service Azure Policy[2] to enforce organizational standards and compliance for the resources that exist. Azure Policy includes built-in policies that you can use or custom policies that allow you to create your own policies. The Azure policies are written in JSON and deployed upon cloud platform Azure.

Creating your own custom policies requires knowledge on how the structure is built and how to use special operators and inbuilt parameters[3]. The combination of using built-in and proprietary policies can complicate the process and create "anarchy" around simple and given requirements or controls. In other words, it is easy to complicate the process so that the effect is not maximized around control or safety.

In this project, I create an "Azure Policy Builder" that democratizes the language and vocabulary of Azure assets to make it visible, simpler, and more unified to a larger audience. The central idea is creating a web-application in which it is possible to easily connect elements to visually create a policy. To receive the resulting Azure Policy, a user only needs to type in his custom values instead of building the entire JSON structure by himself. This is done through click & add. In other words, the user can connect elements according to his needs and then get this out in JSON, which then can be used to set up an own policy in Azure correctly and efficiently.

## 1.1 Customer

The project idea for an "Azure Builder" was given by the company Ombori[4], more specific the security/compliance department. Ombori is a company that develops applications to connect physical stores with e-commerce and digital channels.

## **1.2 Purpose and Goal**

The purpose of this project is to simplify the creation of Azure Policies and to democratize the language and the vocabulary of Azure assets, so they become discoverable for a greater audience.

The goal is to create a web-application that, through using a user-friendly method like click & add or drag & drop, will use elements to visually set-up the structure for the Azure Policy. The result of the creation will be displayed as policy JSON structure and is supposed to be copied and deployed upon Azure Platform.

## **1.3 Limitations**

The main limitation of this project is that it is possible to create only Azure policy definitions. There is no planned support for templates, the creation of groups, or other techniques offered by Microsoft.

## 2 Background

This section covers existing techniques related to this project, different platforms, libraries and languages used.

### 2.1 Similar existing techniques

To the best of my knowledge, currently, there is no work or work in progress on an Azure Policy builder. However, a similar technique exists which utilizes drag&drop for coding as well as other projects which try to make the creation of Azure Policy more efficient. One of them is the original Microsoft Azure Policy which is constantly being developed. Others to mention are Game Maker which has drag&drop function for coding, and Scratch which is utilizing drag&drop to simplify programming.

#### 2.1.1 Game Maker Drag&Drop

Game Maker[5] is an engine for creating your own games, which can utilize the drag&drop function[6] to easily dive into the creation of games and Game Maker's language GML. The similarity to this project is the drag&drop function and how Game Maker uses this functionality to simplify the creation of code.

#### 2.1.2 Scratch

Scratch[7] is a program that lets a consumer create games, animations, and stories. The programming utilizes a drag&drop called Snaps![8]. In short, it is a visual drag and drop programming language with the purpose of introducing computer science to a younger audience. Scratch is a great reference for this project. Even though the target audience is youngsters, it demonstrates a great example of how drag&drop can simplify coding and structure for the language. The program is also highlighting an essential factor, user-friendliness. It is easy to use the program thanks to its intuitive user interface, and this is something significantly important to remember when developing any web-application.

### 2.2 Azure

Microsoft Azure[1] is a cloud-platform developed by Microsoft. It allows users to manage resources and services through their platform. It provides Infrastructure as a Service (IaaS)[9], Platform as a Service (PaaS)[10] and Software as a Service (SaaS)[11]. Azure is a large cloud service provider and has over 60 data centers spread across the regions [12]. Thanks to being so spread out, they can reach out to a larger audience. Azure provides a wide selection of services and platforms to use, one of which is Azure Governance.

## 2.3 Azure Governance

Azure Governace[13] contains the services for governance and management of resources through the use of Azure Policies, Azure Management Groups, Azure Blueprints, Azure Resource Graph and Azure Cost Management and Billing. To get started with Azure Governance, it is essential to know the relations between the different building blocks.

### 2.3.1 Tenant

Tenant[14] is the Top-tier block which the consumer, company, or private person starts off with. Tenant is unique for the consumer, and it is possible for a consumer to have multiple unique tenants.

### 2.3.2 Subscription

Subscriptions[14] open up the possibility to use the Azure cloud services, i.e., IaaS, SaaS and PaaS. It is possible for a consumer to have more than one subscription.

### 2.3.3 Management Group

Management Group[14] is connected to a tenant and is used to group subscriptions. The tenant always starts with a root management group that can not be deleted, only expanded. From the root management group, more management groups can be connected to it. They are used to create a logical structure for the consumer.

### 2.3.4 Resources

Resources[15] are the objects in Azure such as Virtual Machines, databases, etc. Resources are last in the chain for the hierarchy.

### 2.3.5 Resource Group

Resource groups[14] are used to group multiple resources. When resources are grouped, they can be treated as a single object, and all resources within inherit what is assigned to the group.

### 2.3.6 Azure Policy

Azure Policy is a service from Azure that is used to enforce organizational standards and compliance for the resources within Azure. With policies, it is possible to easier control and manage user's resources, for example, to deny the creation of different kinds of virtual machines. Azure provides a lot of pre-built policies that can be used and assigned through Azure, and they can be assigned to different scopes within Azure. These scopes include management group, subscription, resource group, or resources. It is important to know the

hierarchy of these scopes to correctly assign the policy[14][16]. An example is shown on figure 1.

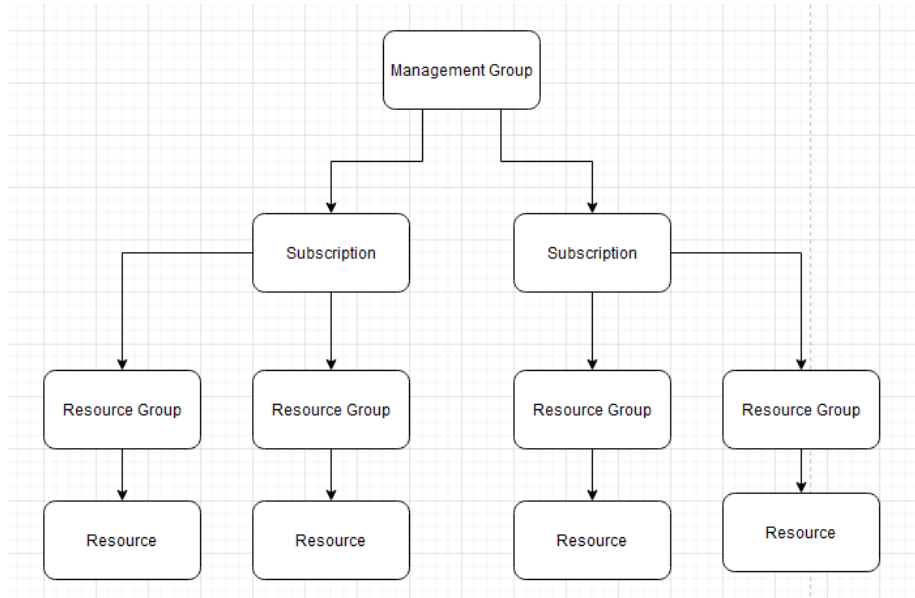


Figure 1: Hierarchy of the scopes available

If you assign a policy to the subscription, all the resource groups and resources within that subscription will be assigned with this policy. Another example is if a resource group is assigned a policy, then only the resources below will be assigned. In other words, everything under the level will be assigned the same policy.

### 2.3.7 Azure Policy Definition Structure

The policies have a pre-defined structure that has to be followed to be successfully created. The policy is written in JSON through an editor such as Visual Studio Code. The policy structure is defined by Microsoft documentations[17] as:

- display name
- description
- mode
- metadata
- parameters
- policy rule
  - logical evaluation
  - effect

The structure mentioned above is to be followed, with some exceptions. Display name, description and metadata are created upon the creation of the policy through the Azure Portal. During the creation of the written JSON policy definition, the only parameters needed are "mode", "parameters" and "policy rule". "mode", "parameter" and "policy rule" in turn have a structure and some set options that have to be chosen.

This structure is also defined in the Microsoft documentation[17]:

Mode:

- indexed or all

Parameters:

- name
- type
- metadata
  - description
  - displayName
  - strongType (Optional)
  - assignPermissions (Optional)
- defaultValue (Optional)
- allowedValues (Optional)

Policy rule:

- if
  - logical operator & condition
- then
  - effect & details



During the project this structure will be analyzed and utilized for the web-application to create the policy definition. An example of how a complete policy definition can look like is shown on figure 2, taken from one of many Azure built-in policy.

```
{
  "parameters": {
    "listOfAllowedSKUs": {
      "type": "Array",
      "metadata": {
        "description": "The list of SKUs that can be specified for storage accounts.",
        "displayName": "Allowed SKUs",
        "strongType": "StorageSKUs"
      }
    }
  },
  "policyRule": {
    "if": {
      "allOf": [
        {
          "field": "type",
          "equals": "Microsoft.Storage/storageAccounts"
        },
        {
          "not": {
            "field": "Microsoft.Storage/storageAccounts/sku.name",
            "in": "[parameters('listOfAllowedSKUs')]"
          }
        }
      ]
    },
    "then": {
      "effect": "Deny"
    }
  }
}
```

Figure 2: Example of Policy Definition

### 2.3.8 Initiatives

Initiatives[14] are used to group up policies. That simplifies management thanks to having a single item instead of multiple. Instead of handling multiple policies and keeping up with updating every single one, it is possible to group them up into a single item. By doing so, the workload is reduced as well as the chance of forgetting a policy.

## 2.4 Software

In this project, CSS, HTML, and the programming language TypeScript are used to create the web-application.

### 2.4.1 TypeScript

TypeScript[18][19] is an open-source language that is built on JavaScript and introduces types. Types are used to describe the shape of an Object, provide documentation and allow validation to check if the written code is working properly. TypeScript is not a new language and transpiles the code into JavaScript. However, it chooses a more performance-wise JavaScript code through the typed TypeScript.

### 2.4.2 Cascading Style Sheets(CSS)

CSS[20] is used to manage style, color, and all other elements representing the visual appearance in web-applications. It is used to make the web-application more visually pleasing for the consumer.

### 2.4.3 HyperText Markup Language(HTML)

HTML[21] defines the structure of the content for a web-application. HTML is a markup language that creates elements. The elements are used to wrap or enclose parts of the content to structure it and make it appear or act in certain manners[21]. CSS is used upon the HTML to style and make it more visually pleasing.

## 2.5 Platforms & Libraries

The platform that is utilized during this project is React. The libraries are drag&drop library called react-dnd-beautiful[22] and react-hook-form for handling input data.

### 2.5.1 React

React is a JavaScript library maintained by Facebook [23]. It is used to create interactive web pages. It is component-based which makes it easy to manage and divide parts of the application. It also has the functionality to easily host one's website through localhost, which can be used to see the application visually during development. React uses the language JavaScriptXML or TypeScript. In this project TypeScript is used along with React.

### 2.5.2 React-dnd-Beautiful

The drag&drop that is used in the project is react-dnd-beautiful[22]. React-dnd-beautiful is a library that needs to be downloaded through npmjs[24]. To use this library, you encase the whole react application in a DragDropContext.

In the DragDropContext, you encase the components in Droppable, and in Droppable you encase the component with Draggable in order for the component to be draggable. Visually it can be seen in figure 3.

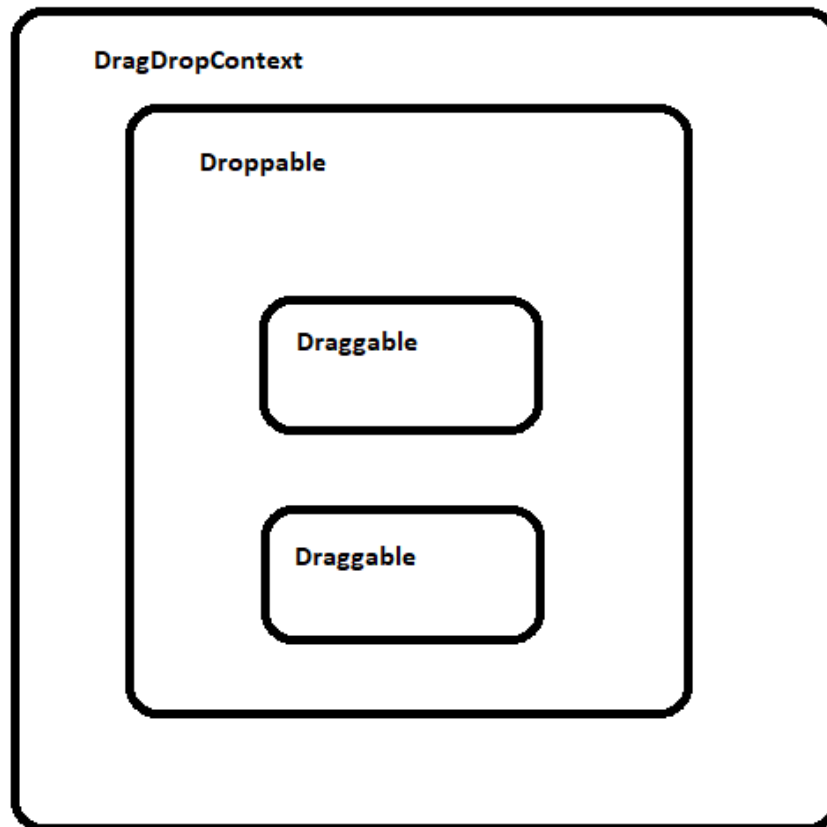


Figure 3: DragDrop Contexts

To summarize the contexts, DragDropContext[25] is encasing the whole application, and from there, you can encase with Droppable[26] & Draggable[27] for the components you wish to have this property. These contexts have properties that need to be fulfilled in order to work properly.

DragDropContext[25] has one required property to work which is onDragEnd. OnDragEnd requires a function where the user customizes how to handle the dragged components. OnDragEnd provides the source and destination of the dragged option. The information from source and destination is enough to pinpoint which component is marked and where the destination of this component ends up.

Droppable[26] has one required property called droppableId, which is the id of the droppable component. It also requires that the children of the Droppable are a function that returns a ReactElement. Droppable provides an object "provided" which is used to identify properties for the Droppable. The first one is "innerRef" which is used to bind the ReactElement to appear on the highest possible DOM node in React. The second one is "placeholder" which is used to create room in the Droppable component if the dragged object is not from the same Droppable component. That is, if a new item occurs in the Droppable component, it increases the size of Droppable to handle new items. Lastly, droppableProps exists within "provided". It is used for styling and lookups. It is essential to have droppableProps applied to the same component that "innerRef" is applied to. Otherwise, when the component is moved it will not recognize which component moved.

Draggable[27] has two required properties, draggableID and index. DraggableID contains the id, i.e., identifier, of the component that should be able to be dragged. Index is the order of the draggable components, showing where in the Droppable list it exists. No index duplicates can exist since it is the identifier for the position of the unique draggable component.

### 2.5.3 React-hook-form

React-hook-form[28] is used to collect input data from a web-application. It is a library containing API's for handling forms. In this project, "useForm" and "useFieldArray" are used. "UseForm" is a custom hook that validates the form using minimal renders for existing inputs. "UseFieldArray" is used to dynamically generate new objects and has functions for adding and deleting. "UseForm" is used to handle all data while "useFieldArray" can be used to generate new objects. React-hook-form can use uncontrolled inputs instead of using controlled inputs. Uncontrolled inputs mean that it is possible to reference input and there is no need to rely on states within react to control what values are being written into the input. In this way, it is possible to simply reference the inputs and let the library keep track of each input. It is also easy to integrate the use of react-hook-form into other User-Interfaces components, which otherwise requires different solutions.

## 2.6 Framework

The development of the application will use a framework called Material-User Interface(MUI)[29]. It is possible to use MUI's components to make web development faster and more stylish. In this framework, this project will utilize components such as Buttons, Radio, Icons, styles and the most important, Tree-View.

### 2.6.1 TreeView

MUI's TreeView[30] is the structure the section policyRule will assume, it is a tree with the possibilities to be expanded with TreeItem. TreeView initialize the tree and TreeItem are items that the tree contains. The reason that TreeView is chosen from the countless other tree components that exist is that it is possible for a TreeItem to contain HTML and to customize the tree content freely.

### 2.6.2 Tabs

Tabs is a component from MUI. It allows navigation between groups that contain content. It is a component that can be used to create an easily navigated menu.

### 2.6.3 Other Components

Buttons, Radio, Icons and styles give the baseline for a stylish and functional component that otherwise would be just a simple component. It is fully customizable and looks great without diving deep into the component structure. All the components are well documented and easy to integrate in a web-application. MUI has a lot of different components, and just by browsing through their components, it is possible to visualize great design ideas for a web-application.



## 3 Methods

### 3.1 Pilot Study

This project is about democratizing and simplifying the creation of Azure policies. To determine how to do this and what to implement, Azure Policies had to be analyzed. Microsoft's documentation was the primary source of information about the structure, assigning, resources, etc. The first questions to be answered were "Why does policy creation need to be simplified?" and "How is it possible?" During my pilot study, it was known that the definition structure for creating a policy has a lot of repeatable JSON, with only different values. Also, the definition structure must be followed according to Microsoft's documentation. In general, to get started with policies, you have to read through a lot of documentation which is time-consuming and hard. Having this in mind, it is obvious that the policy creation process can be simplified by removing tedious repetitions for the structure and by reducing the amount of time needed to understand the JSON structure. To simplify the creation itself, drag&drop can be utilized instead of writing using JSON. It is easier to drag components to compose code blocks instead of writing them from scratch.

With this idea of creating a drag&drop Azure policy builder, the next question came to mind was "Where this application has to be created?". With such a Builder, there is only a need to drag&drop components into a list which is the result. No data needs to be stored since the application should create the policy depending on what the user chooses at the current moment. Thus, a single web-application will be enough. Here React came to mind since it is easy to use and it excels at creating one-page web-applications. React is component-based, and is great for creating applications.

A drag&drop was investigated. It was found out that on node package manager[24], there exist different types of drag&drop components that can be used. In the end, react-dnd-beautiful[22] was chosen since it has a good custom option and natural movements.

### 3.1.1 Testing the project

The testing for this project will be conducted in the UI. The testing will involve steps presented in Table 1.

Test	How to test it	Expected result
Navigating through the possible choices	Try all the options in the application	Natural movement and transition between the choices of the application
That a click on the button displays correct JSON	Trying the component through drag&drop and analyzing that the result is correct	produces right JSON from the choices of the consumer
User-friendliness	Self-analyze the result and sending the application to Ombori for testing of the UI	A good easy to use interface
Creation of the definition through click&add	Testing the creation myself and sending to Ombori, where Azure professionals can test it	Natural movement and correct definition from options
Simplicity of the application	Compare it to writing pure JSON and analyzing the result of testing from Ombori	Easy to understand and not tedious to create policy

Table 1: Testing for the project



## 3.2 Development of the System

The development of the application will delve into how the structure for the components is composed and the steps for the creation of each part, starting with explaining how the policy definition structure will be used for the application; how the application is created and the creation of the components within the application; how the lists are composed and how the click&add is implemented and used on the components; lastly, how the choices are composed into the correct JSON for the policy.

### 3.2.1 Azure Definition Structure for the application

The first step in the project is to structure the components for a code-block that represents the JSON from a policy definition. It is important to analyze the current way of creating the definition, and to restructure it to a simpler form, keeping the original JSON but explaining it in an easier way. There are three important "sections" that define a definition. Those are "mode", "parameter" and "policyRule". "mode" can have only two outcomes, indexed or all. First, "mode" is required and can be included directly upon the canvas with the possible option "indexed" and "all". Second, "parameter" is an optional section and if used, has more properties and can be simplified by visually displaying the properties with inputs for the user. These are listed in 2.3.7. The optional parts exist, but if left empty, is not included in the result. The last, PolicyRule defined in 2.3.7, starts with the required "if", "then" and "effect". The logical operators and conditions should be added upon click on respective button from the menu upon the selected "if" statement, and the same with the choices for the "then" properties. All the values have an input field, with a auto-complete option for the built-in parameters for each property. With the structure laid out, the creation of the application is starting.

### 3.2.2 Creating the application and components

To create the react application, node package manager is used. With the creation of the react application, the foundation is the first to be laid. The foundation is the sections in which the different parts of the applications are supposed to be. Following the initial design shown in figure 4, the sections are a menu that holds all the addable options that exist for "policyRule". A container for "parameter" that contains the form for "parameter" properties. A container for "policyRule" that contains the tree-structure for the possibility to dynamically nest the rule definition. Lastly, the "mode" that is a selection for the user.

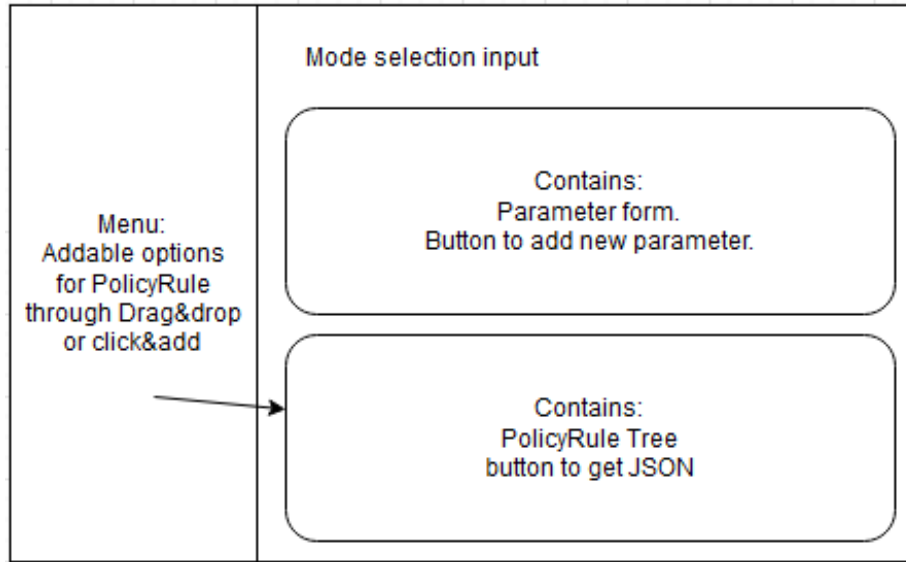


Figure 4: First design layout for web-application. Different sections of application are shown.

To start with the website, these sections shown in figure 4 was defined using HTML. After creating the sections above, their functionality needed to be defined.

### 3.2.3 Parameter functionality

The container for "parameter" functionality included showing the possible properties for the user. For these "parameter" properties inputs was required for the user to define their desired values. The parameter's properties as described in 2.3.7 was defined upon the parameter section. These parameter properties are: "name", "type", "description", "displayName", "strongType", "assign-Permissions", "defaultValue" and "allowedValues". To handle the input data, react-hook-form was used, in this case a specific API from react-hook-form was used called "useFieldArray". The reason why "useFieldArray" was used was that there can exist multiple parameters within a policy definition, and "useFieldArray" makes it possible to simply add a new item through their in-built functions. When adding a new so-called "fieldArray" it duplicates the previous item, which is the parameter form. With this, the functionality to add multiple parameters exists. The inputs must also give the correct JSON data structure. To achieve this using react-hook-form, it is necessary to name the inputs by their respective names shown in figure 5.

```
parameters[${index}]
parameters[${index}].customName
parameters[${index}].customName.name
parameters[${index}].customName.type
parameters[${index}].customName.metadata
parameters[${index}].customName.metadata.description
parameters[${index}].customName.metadata.displayName
parameters[${index}].customName.metadata.strongType
parameters[${index}].customName.metadata.assignPermissions
parameters[${index}].customName.defaultValue
parameters[${index}].customName.allowedValues
```

Figure 5: Naming of Inputs

By naming respective inputs by the correct name, the data forms the correct JSON structure. The indexing in "parameters" is what "useFieldArray" utilizes to create duplicates of the parameter form. The names that are nesting through dots upon other names, create the correct JSON structure.

### 3.2.4 PolicyRule functionality

The container for "policyRule" has to involve handling objects that could be nested and dynamically changed according to the user's self-defined rule. To start of a object was defined, the object which would represent one block for the "policyRule" is shown in figure 6. The object is a classic node, with some added properties to define the options available.

- id: The unique name of the Input.
- previd: The previous Input id.
- secondOption: The second option for the property.
- amountChildren: The amount of children this Input posses.
- icon: The icon displayed next to this Input.
- option: The main property option.
- children: Nesting of the same type, to expand the tree.

```
export type Inputs = {  
  id: string;  
  previd?: string;  
  secondOption?: string;  
  amountChildren?: number;  
  icon: JSX.Element;  
  option: string;  
  children: Inputs[];  
};
```

Figure 6: Input Object

TreeView was used to create the tree from the "Inputs" data. To render this tree structure, the function `renderTree()` (figure 7) was created. `renderTree()` as the name implies is the function that returns the correct tree-structure visually. The function is recursive and in this manner can create the tree. The `TreeItem`'s identifiers are `key` and `nodeId`, whereas `icon` and `label` are what is visually showed for the user. The label uses another function to determine what to render upon the node. The function `findComp()` filters by `option` and `secondOption` and returns the correct HTML element to display upon the canvas. The identifier `nodeId` determines if the node is the one that is selected by the user. There is also an `Icon` and `onIconClick`. The icon renders the icon given, and `onIconClick` is an "onClick" handler that triggers the function `deleteInput()`. `deleteInput()` is a function that removes the node with that id if clicked upon the icon.

```
const renderTree = (treedata: Inputs) => {
  return (
    <TreeItem
      classes={{
        root: treeClasses.root,
        content: treeClasses.content,
        group: treeClasses.group,
        label: treeClasses.labelRoot,
      }}
      key={treedata.id}
      icon={treedata.icon}
      onIconClick={() => deleteInput(treedata.id)}
      nodeId={treedata.id}
      label={findComp(
        treedata.option,
        treedata.id,
        treedata.previd,
        treedata.secondOption
      )}
    >
    {Array.isArray(treedata.children)
      ? treedata.children.map((treedata) => renderTree(treedata))
      : null}
    </TreeItem>
  );
};
```

Figure 7: RenderTree, recursive render function

TreeView is the the initialiser used to create the tree. Inside TreeView, the function that was explained `renderTree()` is used. The parameter "policy", used in `renderTree()`, is the start data of type "Inputs" containing the required policy rule properties mentioned in 2.3.7. A function helper `addInput()` also exists, it adds a new "Inputs" upon the selected node "children" property. To visually change the render for the user when adding or deleting "Inputs", the data of the objects are contained within states. "Policy" is the local state, so whenever a change occurs from deleting or adding, it will run the `renderTree(policy)` function, which triggers a change upon the canvas. This `addInput()` function is utilized in the menu-section.

```
<TreeView
  onNodeSelect={handleSelect}
  expanded={nodeIds}
  className={treeClasses.TreeViewRoot}
>
  {renderTree(policy)}
</TreeView>
```

Figure 8: The structure of TreeView, with selection of the items enabled.

### 3.2.5 Drag&Drop or Click&Add

The drag&drop (2.5.2) was tested at the start of this project to determine if it fits the web-application. However, when "parameters" and "policyRule" were completed and it was time to implement drag&drop, it felt clunky with the use-case of "policyRule". It did not have the freedom to connect whatever a user wants since it is important to follow a certain structure. Usually, when one wants to add something, it would be faster to just click the option instead of dragging it. Hence, the final decision was to use click&add instead. This would give more precision during add-operation and is faster when adding multiple options. The only upside drag&drop would have was to be able to re-order the options, however, this is not necessary. The implementation was to instead of using drag&drop, have buttons at the menu. The user selects the node on the canvas and adds the desired condition/property through a button click.

### 3.2.6 Menu functionality

With the completion of sections "parameter" and "policyRule", it was time to create the menu which would expand the tree for "policyRule"-section. The menu that has the reserved spot at the left of the web-application was created by using "Tabs" component from MUI. It is a simple menu component that creates clickable "tabs".

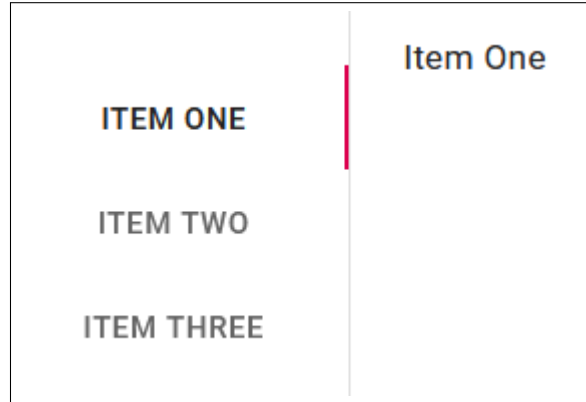


Figure 9: An example of the look of Tabs Vertical

The options we need for the menu "Tabs" are:

- operators
- value/field Conditions
- append
- auditIfNotExists
- audit/deny
- modify

The content within the "Tabs" was the corresponding options that could be used to create the JSON structure for "policyRule". It consisted of MUI buttons with "onClick" handlers. When clicked, it sends the corresponding option to the function `addInput()`, where it checks what the user selected and creates new "Inputs" in the "children" property for this "Inputs".

### 3.2.7 The resulting JSON data

With the completion of the sections and the setup for react-hook-form, it was time to display the result JSON data. However, the data that was gathered throughout the web-application had the incorrect JSON structure that Azure requires. Before printing the data to the user, it was necessary to handle it and

fix the incorrect structure. The incorrect data was "parameters" and some of the inputs. React-hook-form gave out parameters as an array, whereas in the Azure structure it is simply an object containing the different created parameters. Another issue in parameters is that the id name is the written name that a user defines. To fix both of these issues, the array had to be iterated and added upon a newly created Object. The new Object replaced the parameter data. A new Object had to be created because Objects are immutable, meaning it can not dynamically change the id name. The second issue was that some inputs required the output values to be arrays, inputs output string values, and this had to be changed. In the inputs, it was necessary to check if this is an array value-option and if it is, creates an array for the values inputted by the user instead. Finally, with the correct structure, all that was needed was to show this JSON data to the user. Instead of displaying the JSON data upon the web application, a pop-up was used. With the use of a pop-up, it is easier to copy the data directly and it does not take up an unnecessary place in the application. The pop-up used was from MUI and is called "modal".

### **3.2.8 Design**

Once the web-application was completed, styling needs to be applied. With the use of MUI some standard styling was already set on components such as buttons and inputs. But tweaks were needed since the sizes were too big and the containers for the sections were not visually pleasing to look at. The whole application design was revised and styles were added to all components and containers. The progression of the design for the web-application can be seen in Appendix B.

### **3.2.9 Feedback changes**

Feedback was received from the supervisor from Ombori, where it became clear that the design had flaws. The menu was revised, and instead of using "Tabs" for the menu, it was switched to TreeView. Color-coding the buttons were added to make the web-application more user-friendly. Now, blue buttons depict "add" and red - "deleted" whereas all buttons had one color in the previous option. Icons were added on the buttons to indicate their purpose, where a "+" means adding and trashcan - deleting. In the feedback, it was also mentioned a lack of information given to the user. To solve this, a hover action was implemented. When hovering over an i-icon, it gives information about the corresponding property. The design was slightly altered, so instead of having horizontal containers for "parameters" and "policyRule", it was changed to vertical containers that were more fitting since "policyRule" contains a tree-structure.



## 4 Results

### 4.1 Website & Website Sections

The result of this project is a website that produces the JSON structure from a user's selection and inputs on the canvas through click & add. The website contains 4 sections: the left-sided menu, the policyRule canvas, the parameter canvas, and a little section which includes the 'mode' and 'GET JSON' button.

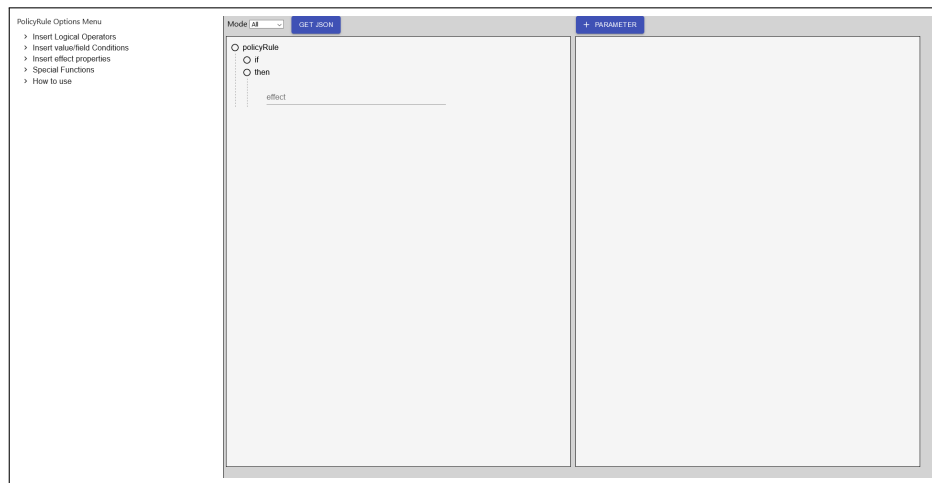


Figure 10: Website appearance

#### 4.1.1 Parameters

Parameters are an optional part, hence, it is empty at the beginning. A button exists to 'Add Parameter', that adds one parameter form. The form requires values from the user.



Figure 11: Empty Parameter

When a new parameter is added, a form is created in which the user can input the information for the parameter. The required fields are marked with a red star, the other fields are optional as shown below in figure 12. A button exists at the end of the parameter, 'REMOVE PARAMETER', which removes the parameter. It is possible to add more parameters, the newly added parameter will add the same form below the latest added parameter.

Figure 12: One Parameter

An icon in the front of each parameter property name, when hovered with the mouse, displays a pop-up information about what is expected and what this property means, as shown below in figure 13.

Figure 13: Hover Information

#### 4.1.2 Menu

The menu consist of a tree, with the options that PolicyRule can contain. The tree is of type TreeView from MUI. The items within the tree are buttons that add to the selected node from the PolicyRule canvas. The menu contains, as seen in figure 14, all the options required for a policy rule and optional properties for the effect.

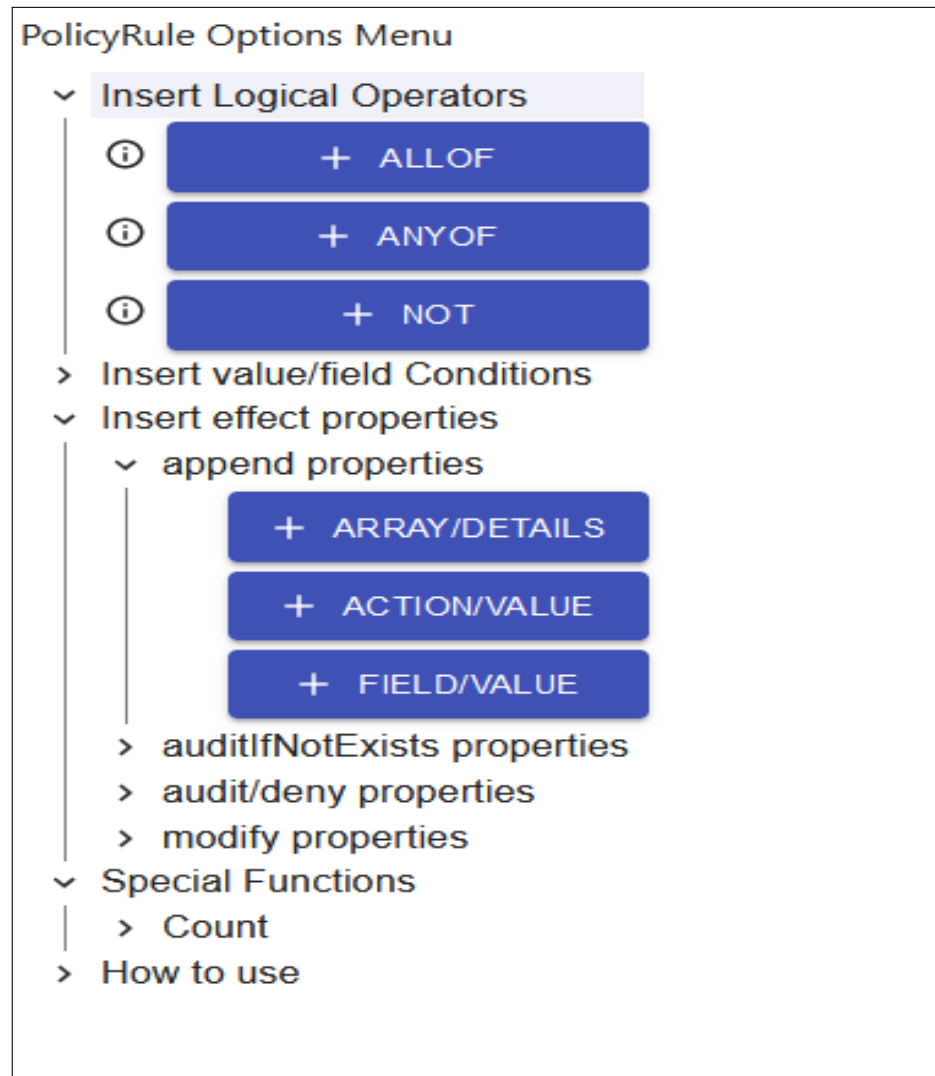


Figure 14: Menu

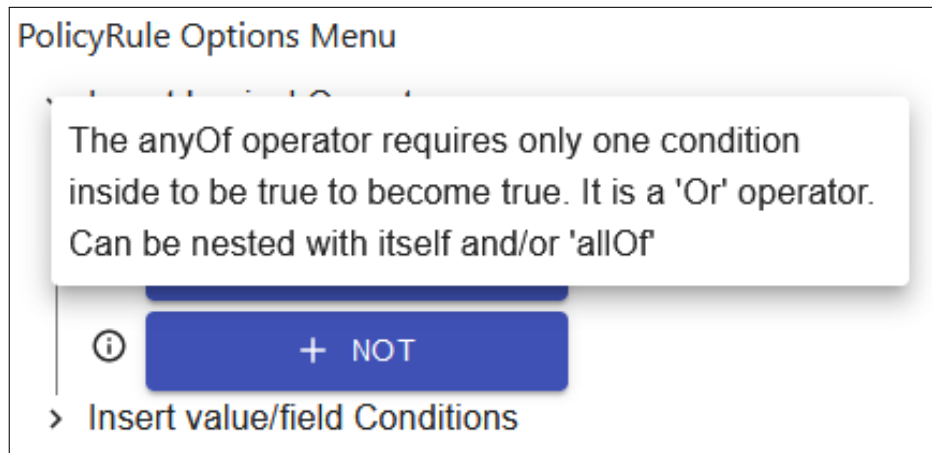


Figure 15: Hover Event

Information regarding the options that can be added, can be shown through the icon on the left of the buttons. The information contains some basic information about what this option does. It is a pop-up triggered by hovering over the icon.

#### 4.1.3 Policy Rule

The policyRule contains a canvas with the tree-structure. In the beginning, the required parts that policyRule consists of, which are 'if', 'then' and 'then.effect', is shown. When an option is selected, it is lighted up. Once the option is selected, it is possible to use the menu buttons to add operators, conditions and effect properties. PolicyRule can be nested and to see what contains what, a dotted line indicates which option is under it.

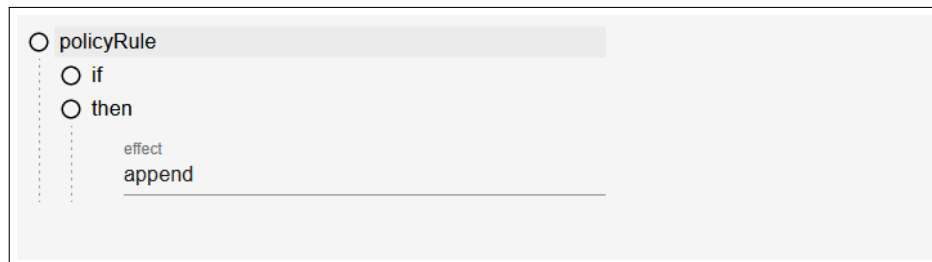


Figure 16: PolicyRule Canvas

For example, if selecting 'if' and adding an 'allOf' it will create a field with 'allOf'. After selecting 'allOf', it is possible to add conditions such as 'equals'. This will create inputs with either 'field' or 'value' and the condition 'equals'. It is possible to nest operators, so if the user selects 'allOf' again and adds operator 'anyOf', another field named 'anyOf' is created. By selecting 'anyOf' and adding two conditions, 'notEquals' and 'contains', it will create the inputs. This is shown in figure 17. The inputs seen next to the added properties on the canvas expect the user to fill them in.

```
graph TD
    policyRule((policyRule)) --> if((if))
    policyRule --> then((then))
    if --> allOf[allOf]
    if --> anyOf[anyOf]
    allOf --> field1[field]
    allOf --> equals[equals]
    anyOf --> field2[field]
    anyOf --> notEquals[notEquals]
    anyOf --> contains[contains]
    then --> effect[effect]
    effect --> append[append]
```

Figure 17: Adding Options

In figure 17, the added options have an icon of a trashcan. By clicking the icon, added options are deleted. By deleting an option, it will also delete all the added options nested within it.

#### 4.1.4 Mode & Get JSON

Lastly, one more mini-section exists, which includes a simple select and a button 'GET JSON'. The selection gives the user two options, either 'All' or 'Indexed'. This sets what 'mode' is. The last button is 'GET JSON', positioned next to 'mode'. It triggers a pop-up with the structured Azure Policy JSON for the options chosen from policyRule, mode and parameter.



Figure 18: 'Mode' & 'GET JSON' buttons

For example, if we take the structure shown on the figure 17, add one parameter and some fake-values, and push the 'GET JSON' button, we receive JSON structure shown on the figure 19.

A screenshot of a web application's JSON viewer. The viewer has a close button (X) in the top left corner. The JSON data is displayed in a monospaced font with syntax highlighting. The structure is a nested object representing a policy rule. It includes a 'mode' field set to 'All', a 'policyRule' object with an 'if' condition (using 'allOf' and 'anyOf' for field checks) and a 'then' effect (calling a function 'parameters' with 'effect' as an argument). A 'parameters' object is also defined at the bottom, specifying the 'effect' as a 'String' with a description and display name.

```
{
  "mode": "All",
  "policyRule": {
    "if": {
      "allOf": [
        {
          "field": "someValue",
          "equals": "true"
        },
        {
          "anyOf": [
            {
              "field": "AnotherValue",
              "notEquals": "false"
            },
            {
              "field": "ThirdValue",
              "contains": "someStorage"
            }
          ]
        }
      ]
    },
    "then": {
      "effect": "[parameters('effect')]"
    }
  },
  "parameters": {
    "effect": {
      "type": "String",
      "metadata": {
        "description": "This effect will do something",
        "displayName": "Effect"
      }
    }
  }
}
```

Figure 19: Resulting JSON data



## 4.2 Test Results

Testing of the application was done by 5 selected people. While they went through the website, I was watching them silently to see if they could grasp how to use the website. From this information, it was concluded that the application could become more user-friendly., e.g., the website required more information on how to get started. The test result was adequate to the expected result. The result can be seen in Table 2.

Test	How to test it	Expected result	Result
Navigating through the possible choices.	Try all the options in the application.	Natural movement and transition between the choices of the application.	Works as expected.
That a click on the button displays correct JSON	Trying the component through click&add and analyzing that the result is correct.	produces right JSON from the choices of the consumer.	Works as expected.
User-friendliness	Self-analyze the result and sending the application to Ombori for testing of the UI.	A easy to use interface.	Not fully fulfilled, application can become more user-friendly.
Creation of the definition through click&add	Testing the creation myself and sending to Ombori, where Azure professionals can test it.	Natural movement and correct definition from options.	The options visually shows correct upon the canvas.
Simplicity of the application	Compare it to writing pure JSON and analyzing the result of testing from Ombori	Easy to understand and not tedious to create policy.	Application not as simple as expected. Compared to writing JSON the result application is not simpler.

Table 2: Test Result



## 5 Discussion

This project aimed to create an application that would simplify the creation of Azure Policy definitions and can assign the result directly upon Azure Portal. The focus idea of how to achieve this, grew from analyzing the current way of creating Azure Policy definition. The project resulted in removing the need to construct JSON structure when defining the policy definition. Expanding the policy definition is done by click&add and every property that expects a value from the user has an input bar.

### 5.1 Result compared to goals

The goals for this project were:

1. Create an application that produces the JSON policy definitions.
2. Using a user-friendly method such as drag&drop or click&add, create the policy definition through the web-application.
3. Simplifying the creation of the policy definition process.

1. The result became a web-application that does produce correct JSON policy from the user's options. However, it does not contain everything Azure can utilize, for example, special functions and one optional effect property "ifNot-DeployExists". The reason for this is that the focus was on making the website more user-friendly since it requires inputs from the user. The usability and how effective it can be depends a lot on the design and information given to the user. By focusing on user-friendliness, the web-application became more understanding and navigation became more logical.

2. Drag&drop was revised and replaced by click&add. During development with drag&drop it became an annoyance to drag and nest items, if misplaced it would require to re-drag it and could potentially make the user add or nest it upon something that was not intended. Clicking on the selected options and adding through buttons, gives more precise adding conditions and is faster if the need arises to add multiple options upon the same selected option. Comparing drag&drop with click&add, there was only one reason why drag&drop would be chosen, which is reordering within the canvas. However, this is not something that is relevant to this project and gives no benefit. Having all this in mind, click&add was chosen for the final design.

3. The web-application did not achieve simplifying the current creation process of Azure policy definitions. However, it did give an alternative way of creating these. The reason why the application did not achieve this goal is that it lacks a trait that would give the user an edge for using the result web-application. The resulting web-application gives to beginners a slight advantage since all the information can be found within the same page where the policy is created. However, it comes with downsides, that is, not supporting other helpful extensions for receiving their resources. It does not have a better auto-complete

for options and code-based gives more freedom to dynamically change mistakes with wrong chosen conditions, property or types. In most cases, users creating the policy definition have a background in programming, which means that they are more comfortable with code-based creation. Simplifying the creation of policy definition is difficult due to that the code-based creation itself is pretty simple. The difficult part rests more on the intake of information regarding what resources to use and how the policy affects these resources within Azure, and how to assign and manage these policies.

## 5.2 Flaws in the project

One big weakness during this project is the lack of experience in creating Azure policies definition and managing resources within Azure. Because of the lack of experience, a lot of time was spent on understanding the policy definition structure as well as understanding how it affects resources in the Azure environment. It seems that with wider prior knowledge on handling resources within Azure, the mindset on how to simplify the creation of Azure policies would have been different from the current result.

A lot of time was spent on the drag&drop and attempts to simplify the creation process with it. I got stuck for quite some time, trying to find a way to combine them to make it plausible. Instead of continually trying to use drag&drop for this particular use-case, I should have dropped it earlier and implemented the basics to realize that click&add would be a better fit. This would have given more time to implement more functionality and produce a higher-quality web-application.

## 5.3 Strengths in the project

Some pros of this project result have been:

- The resulting web-application is a good basis for further development.
- The tree structure for "policyRule" is visually pleasing and easy to follow, it accurately shows the nesting in a simpler way than the current code-based creation.
- The menu is easy to use and navigate through. Keeping all the required properties under each section with the corresponding information makes the creation process more streamlined.

## 5.4 Assessment and evaluation of the project

### 5.4.1 Implementation

The implementation followed the time plan that was created at the beginning of the project (Appendix A). However, changes appeared frequently during the development of the web-application. This refers to the design and use

of drag&drop. The design during development was frequently changed and tweaked. Time for designing the web-application was not accounted for because of the lack of experience in developing web-applications.

#### **5.4.2 Technical Solution**

The technical solution for this project required a thorough understanding of Azure Policy. The information that revolved around it was enormous, and understanding of just creation of Azure policy definition was not enough. It was also important to know the whole process - from creation to assigning, and how this affected the environment. Without this information, it was impossible to find a reasonable idea for simplifying.

Typescript was used to create the whole web-application. It was important to quickly understand how new components from different frameworks can be used to develop the web-application. The same applies to the different functions that can be utilized in libraries to adapt the functionality for the desired outcome. Understanding of those components and functions implies an intensive reading of countless documentation and trying them out to determine if they can be used for the development.

### **5.5 Social requirements for technical product development**

#### **5.5.1 Economical**

Azure Policy is free to use for the resources owned in Azure. The cost is laid upon the resources that are being used. Azure Policy is used to manage the resources in Azure environment, hence, it costs nothing to utilize it.

The policies can govern the spending of the resources and shutdown these if they drain too much money. They can also be used to deny future creation of expensive resources that employers should not deploy. This project has no cost, the only cost can be the hosting. However, since the web-application does not have many users, the hosting is free.

#### **5.5.2 Environmental**

Azure policies are hosted on the Microsoft data center. The environmental effects include carbon, hardware such as servers, cooling these servers, waste management and other aspects. Microsoft has 4 main areas in which they continuously improve their environmental impact on the world. These include carbon, ecosystems, water and waste. It is important as a user, when using Microsoft Azure, to know that when using their product how this affects the environment. Microsoft is carbon neutral since 2012 and aims to become carbon negative by 2030[31].

The web-applications environmental impact is the server which the application

is hosted upon. To keep the application up at all times, requires a server which is handled by either yourself or a service provider. This includes maintenance of hardware and in turn waste management.

### **5.5.3 Security**

Azure Policy involves enforcing organizational standards and assessing compliance at-scale. The focus is laid upon security within the environment. No personal data are available in the open access, and the policies that are applied to the environment are encrypted. There are also security policies that let the administrator define policies that follow the company's security needs. By defining security policies, it is possible to define the sensitivity of the data across all subscriptions. During this project, security was not the main focus of the application since it serves as a creation-type application. No data is saved anywhere. However, if future development occurs that instills a log-in, security would be an important topic and would need encryption and a solid database that needs to follow GDPR.

## 6 Conclusion

The goals of the project were achieved, namely, a web-application was created that produces JSON policy definition depending on users choices. Simplification was not fully achieved. However, it is worth mentioning that compared to how easy the current process is, it is hard to simplify it further without making it more complex. The resulting application did not fully achieve a simplification goal but serves as a basis for further development and produced an alternative creation method that removed the need to structure the JSON policy definition.

### 6.1 Improvements

The result application involves a good basis to enhance user experience. Improvements on the web-application are to strengthen the user experience for the policy creation process and to involve all the possible options/functions a policy definition rule can handle. Future work can involve expanding the functionality by also being able to create policy initiatives and involve extensions that can fetch resources within the Azure environment.

## 6.2 Validation from Ombori

The supervisor from Ombori, Zsolt Háló, was kind to give his thoughts regarding the result for this project:

”We at Ombori work a lot with clients where we must investigate their internal security and compliance requirements. Understanding compliance regulations is only a tiny part of the problem when you are trying to assess an environment.

Each environment comes with its own nuances’ about how to evaluate security posture. We challenged Kevin with a non-trivial task of democratizing Azure Policy’s syntax and semantics. At the time of writing, the intricate structure of the language (Azure Policy) allows little to no insight into the breadth and width of possibilities that the language can express.

The task was to implement some clever/creative solution that helps a naive user with a limited understanding of Azure to formulate basic Azure Policy documents and to get a have a good understanding of what’s possible to express using the building blocks of the language provides.

Kevin implemented his solution fully based on his creative genius, often challenging the shortsighted ideas described in the original task.

The outcome of the thesis is a handy tool that allows for the convenient exploration of the Azure Policy language and the Azure compliance automation landscape.

We were happy to host Kevin as a thesis student, and we wish him a fruitful and exciting journey throughout his career.

Best, Zsolt Halo - Managing Director of Ombori SecOps AB”



## References

- [1] Microsoft Azure. What is azure? <https://azure.microsoft.com/en-us/overview/what-is-azure/>. Accessed: 2021-02-24.
- [2] Microsoft Azure Documentation, Contributors: David Coulter, Kurt Furbush, Robert Lyon, JH, John Downs, Camille Marvin, Alma Jenks, and Ron Balter. What is azure policy? <https://docs.microsoft.com/en-us/azure/governance/policy/overview>. Accessed: 2021-02-24.
- [3] Microsoft Azure Documentation, Contributors: David Coulter, and Bill Anderson. Tutorial: Create a custom policy definition. <https://docs.microsoft.com/en-us/azure/governance/policy/tutorials/create-custom-policy-definition>, May 2020. Accessed: 2021-02-24.
- [4] Ombori. Ombori homepage. <https://ombori.com/>. Accessed: 2021-02-24.
- [5] YoYo Games. Gamemaker yoyo games. <https://www.yoyogames.com/gamemaker>. Accessed: 2021-02-24.
- [6] YoYo Games. Drag and drop overview. [https://docs2.yoyogames.com/source/\\_build/3\\_scripting/1\\_drag\\_and\\_drop\\_overview/index.html](https://docs2.yoyogames.com/source/_build/3_scripting/1_drag_and_drop_overview/index.html). Accessed: 2021-02-24.
- [7] Scratch. Scratch. <https://scratch.mit.edu/>. Accessed: 2021-02-24.
- [8] Snap! Snap! build your own blocks. <https://snap.berkeley.edu/about>. Accessed: 2021-02-24.
- [9] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.
- [10] Dimpi Rani and Rajiv Kumar Ranjan. A comparative study of saas, paas and iaas in cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(6), 2014.
- [11] S Satyanarayana. Cloud computing: Saas. *Computer Sciences and Telecommunications*, (4):76–79, 2012.
- [12] Microsoft Infrastruktur. Global infrastruktur med azure. <https://azure.microsoft.com/sv-se/global-infrastruktur/>. Accessed: 2021-02-24.
- [13] Microsoft Azure Governance. Azure governance. <https://azure.microsoft.com/sv-se/solutions/governance/>. Accessed: 2021-02-24.
- [14] Peter De Tender, David Rendon, and Samuel Erskine. Pro azure governance and security. *Berkeley, CA: Apress*, 2019.

- [15] Contributor: DCtheGeek Microsoft Resource Graph Documentation. Azure resource graph table and resource type reference. <https://docs.microsoft.com/en-us/azure/governance/resource-graph/reference/supported-tables-resources#resources>. Accessed: 2021-02-24.
- [16] Microsoft ARM. What is azure resource manager? <https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview#understand-scope>. Accessed: 2021-02-24.
- [17] Microsoft Azure Documentation, Contributors: David Coulter, Elad Perets, PRMerger15, Chris Eggert, Alma Jenks, M. Baldwin, PRMerger6, Bharath Nimmala, stratusjerry, Tomáš Bohuněk, Bill Anderson, Lyon Till, Tom FitzMacken, Camille Marvin, Richard Burrs, PRMerger13, DanRawlings-MSFT, and huypub. Azure policy definition structure. <https://docs.microsoft.com/en-us/azure/governance/policy/concepts/definition-structure>. Accessed: 2021-02-24.
- [18] Gavin Bierman, Martín Abadi, and Mads Torgersen. Understanding typescript. In *European Conference on Object-Oriented Programming*, pages 257–281. Springer, 2014.
- [19] Typescript. <https://www.typescriptlang.org/>. accessed 10 April, 2021.
- [20] Cascading style sheets home page. <https://www.w3.org/Style/CSS/>. Accessed: 2021-02-24.
- [21] Html. <https://developer.mozilla.org/sv-SE/docs/Web/HTML>. Accessed: 2021-02-24.
- [22] react-dnd-beautiful. <https://github.com/atlassian/react-beautiful-dnd>. Accessed: 2021-02-26.
- [23] React homepage. <https://reactjs.org/>. Accessed: 2021-02-18.
- [24] Npmjs homepage. <https://www.npmjs.com/>. Accessed: 2021-02-18.
- [25] Erik Beuschau Contributors: Alex Reardon, Tim Haywood. Dragdrop-context. <https://github.com/atlassian/react-beautiful-dnd/blob/HEAD/docs/api/drag-drop-context.md>. Accessed: 2021-02-26.
- [26] Sebastian Aigner Contributors: Alex Reardon. Droppable. <https://github.com/atlassian/react-beautiful-dnd/blob/HEAD/docs/api/droppable.md>. Accessed: 2021-02-26.
- [27] Alex Reardon. Draggable. <https://github.com/atlassian/react-beautiful-dnd/blob/HEAD/docs/api/draggable.md>. Accessed: 2021-02-26.





- [28] react-hook-form. <https://react-hook-form.com/>. accessed 10 April, 2021.
- [29] Material-ui. <https://material-ui.com/>. accessed 10 April, 2021.
- [30] Treeview api. <https://material-ui.com/components/tree-view/>. accessed 10 April, 2021.
- [31] Environmental sustainability. [https://www.microsoft.com/en-us/corporate-responsibility/sustainability?activetab=pivot\\_1:primaryr6](https://www.microsoft.com/en-us/corporate-responsibility/sustainability?activetab=pivot_1:primaryr6). accessed 10 April, 2021.



## 7 Appendices

### 7.1 Appendix A

# Time Plan

 Name	 Assign	 Date	 Status
<u>Pilot Study</u>		@Jan 27, 2021 → Feb 14, 2021	
<u>Setup React(website)</u>		@Feb 14, 2021 → Feb 19, 2021	
<u>Tests on Drag&amp;Drop</u>		@Feb 15, 2021 → Feb 25, 2021	
<u>Program the Canvas for Drag&amp;Drop</u>		@Feb 21, 2021 → Mar 3, 2021	
<u>Structure defined for Policies</u>		@Feb 27, 2021 → Mar 13, 2021	
<u>Azure "mode" all components</u>		@Mar 8, 2021 → Mar 15, 2021	
<u>Test for one layer to produce correct JSON</u>		@Mar 12, 2021 → Mar 16, 2021	
<u>Azure "parameters" all components</u>		@Mar 15, 2021 → Apr 3, 2021	
<u>Azure "PolicyRule" all components</u>		@Mar 15, 2021 → Apr 3, 2021	
<u>Implement Drag&amp;drop or buttons to add new inputs</u>		@Apr 4, 2021 → Apr 11, 2021	
<u>Implement Custom values for each Azure component which can include custom parameters</u>		@Apr 7, 2021 → Apr 14, 2021	
<u>Connect layers for each Azure component</u>		@Apr 8, 2021 → Apr 15, 2021	
<u>Final tests</u>		@Apr 15, 2021 → Apr 22, 2021	
<u>Milestones:</u>			
<u>Projectplan complete and sent in</u>		@Jan 24, 2021 → Feb 2, 2021	
<u>Projectplan Seminar</u>		@Feb 1, 2021 → Feb 5, 2021	
<u>Half-time Report</u>		@Feb 8, 2021 → Mar 10, 2021	
<u>Half-time Seminar</u>		@Mar 15, 2021 → Mar 19, 2021	
<u>Preliminar final report</u>		@May 2, 2021 → May 10, 2021	
<u>Examination and final seminar</u>		@May 17, 2021 → May 25, 2021	
<u>utExpo</u>		@May 31, 2021 → Jun 6, 2021	
<u>Examination Opportunity 2</u>		@Aug 30, 2021 → Sep 5, 2021	
<u>Untitled</u>			

## 7.2 Appendix B

First design:

Mode indexed

Parameter

name of parameter

type string

metadata {

description displayName strongType assignPermissions defaultValue allowedValues

add anyOf

add allOf

add not

add equals

PolicyRule

if

allOf

"field"

"equals"

"field"

"equals"

then

watch

Submit Query

{ "mode": "indexed", "parameter": { "customName": { "type": "string", "metadata": { "description": "", "displayName": "", "strongType": "", "assignPermissions": "" }, "defaultValue": "", "allowedValues": "" } }, "PolicyRule": { "IF": { "allOf": { "2equals": "", "2equals1": "", "3equals": "", "3equals1": "" }, "then": "", "allOf": "" } }

Second design:

OPERATORS

FIELD AND CONDITIONS

VALUE AND CONDITIONS

APPEND

AUDIT/NOT EXISTS

AUDIT/DENY

MODIFY

DEPLOY/NOT EXISTS

COUNT FUNCTION

HOW TO USE

ADD DETAILS

ADD TYPE

ADD NAME

ADD EXISTENCE CONDITION

Mode indexed

Add Parameter

DELETE SELECTED NODE

policyRule

if

then

effect

deny

Submit Query



Third design:

Operators

+ ALL OF

+ ANY OF

+ NOT

Conditions

Insert effect options

Special Functions

How to use

DELETE SELECTED NODE

Mode | All

GET JSON

+ PARAMETER

policyRule

if

then

effect

audit

Parameter: 1

name of parameter:

type | String

metadata

description:

displayName:

strongType:

assignPermissions:

defaultValue:

allowedValues:

REMOVE PARAMETER

Kevin Brandhild



Besöksadress: Kristian IV:s väg 3  
Postadress: Box 823, 301 18 Halmstad  
Telefon: 035-16 71 00  
E-mail: [registrator@hh.se](mailto:registrator@hh.se)  
[www.hh.se](http://www.hh.se)