System for Cross-Domain Identity Management

Master of Science in Technology thesis
University of Turku
Department of Future Technologies
Security of Networked Systems
June 2020
Roni Ronkainen

Supervisors:
Antti Hakkala
Seppo Virtanen

UNIVERSITY OF TURKU
Department of Future Technologies

RONI RONKAINEN: System for Cross-Domain Identity Management

Type of thesis, 84 p., 83 app. p.
Security of Networked Systems
June 2020

System for Cross-Domain Identity Management (SCIM) was first introduced in 2011 to provide a specification for cloud-based identity management. The identity and access management (IAM) sector is growing rapidly and SCIM aims to meet the needs. SCIM aims to bring a simple, easy and cheap way of managing identities in cloud-based applications, by providing clear protocols and schemas. The protocol and core schema make up the backbone of the specification. SCIM enables developers to choose the implementation methods by themselves which helps with keeping the security because the specification does not rely on one authentication method. The SCIM specifications have been updated twice from 1.0 to 1.1 in 2012 and then to version 2.0 in 2015. The latest version will be used to make a working SCIM application to see what it does.
To understand SCIM better and why it was invented the thesis will contain information about IAM and services around it. The core schema and protocol of the SCIM specification will also be looked at and explaining them more understandably. The project will gather all the information gained and show what the SCIM is in practice.

Keywords: SCIM, IAM, Access Control, Identity Management, Protocol, Schema, .Net, Angular, Cloud-based, Database

Contents

List of Figures **iv**

1 Introduction **1**

2 Identity and Access Management **3**

2.1 Components of Identity and Access Management 4

2.2 IAM Lifecycle 5

2.2.1 Provisioning 6

2.2.2 Authentication and Authorization 8

2.2.3 End of IAM Lifecycle 12

2.3 Identity Management as a Service 13

2.4 System for Cross-Domain Identity Management 14

3 SCIM 2.0 Core Schema **16**

3.1 SCIM Schema 18

3.1.1 Attribute Characteristics 18

3.2 SCIM Model 22

3.3 Users 23

3.3.1 Singular Attributes 23

3.3.2 Multi-Valued Attributes 25

3.4 Groups 28

3.5 Enterprise User 29
3.6 Service Provider Configuration Schema 30
3.7 ResourceType Schema 32
3.8 Schemas 33
3.9 Summary of the SCIM Core Schema 34

4 SCIM Protocol **35**
4.1 Authentication and Authorization 36
4.2 Endpoints 36
4.3 Creating Resources 39
4.4 Retreiving Resources 40
4.5 Modifying Resources 42
4.6 Deleting Resources 47
4.7 Additional operations and endpoints 48
4.8 Service Provider Configuration Endpoints 50
4.9 Summary of the SCIM Protocol 50

5 Project SCIM **52**
5.1 Scoping of the project 53
5.2 The Back-End 56
5.2.1 Database Context and Models 56
5.2.2 Controllers 61
5.2.3 Improvements 64
5.3 Front-End 65
5.3.1 User Pages 66
5.3.2 Group Pages 70
5.3.3 Improvements 72
5.4 Testing 72

5.5 Observations 74

6 Conclusion **79**

References **81**

List of Figures

2.1 Example of an IAM model 6
2.2 OpenID Connect concept 11
2.3 Cross-domain concept 15

3.1 SCIM Model 22

4.1 HTTP status codes used in SCIM 48
4.2 The SCIM status 400 error messages. 49

5.1 Model of the SCIM project 53
5.2 Azure AD user mappings 54
5.3 Azure Ad Group mappings 54
5.4 Source & Target Model 55
5.5 Database context of the source 59
5.6 The models 60
5.7 HTTP POST Workflow 63
5.8 Dashboard view in the UI 65
5.9 Userlist view in the UI 66
5.10 Full user page 68
5.11 Add user page 69
5.12 GroupList 70
5.13 Full group page 71

5.14 Add group page 71
5.15 Test connection 77
5.16 Provisioned 77
5.17 Provisioning logs 78

Chapter 1

Introduction

System for Cross-Domain Identity Management (SCIM) is a specification provided for identity management in cloud-based applications. The framework provides guidelines on how identity data should be transferred between systems. The SCIM specification is divided into two documents, the core schema, and the protocol. The core schema identifies how SCIM data should be modeled and the protocol part specifies the way on how the communication should happen. SCIM aims to make moving users to and from the cloud fast, cheap and easy [1]. [2, 3]

The thesis tries to answer what the SCIM specification holds inside it. The other goal is to create a testbed for testing other applications SCIM compatibility. This is achieved by looking at the SCIM Core Schema [2] and Protocol [3] to better understand what the specification is about. The testbed will consist of two applications that send and receive data between each other in SCIM format. One application is the service provider and the other is the client. The applications are tested so they work as one unit using a user interface and the client-side is additionally tested with Microsoft Azure Active Directory [4] to confirm that the system works.

The theory part will be put into practice with a SCIM project that will include two different applications communicating with SCIM messages. SCIM is meant to be an identity management system and that is why the first chapter will explain what identity

and access management is and what it aims to do. Chapters three and four will be about the SCIM core schema and protocol respectively. The fifth chapter will be about the project and how it was built and tested to meet the SCIM specifications. The conclusion chapter will summarize the project and what are the good and bad things about SCIM. The aim is to give the reader a complete package about SCIM in an understandable way.

The project was made to work as a testbed for an application that takes advantage of the SCIM specification but does not fully comply with it. The idea is to have a pure SCIM application that can test whether the data flow in the existing application works as wanted. With the project having a service provider and client version the testing can be done both with sending and receiving the data. In Chapter Five, the project will be connected to a verified SCIM service provider and tested that it works.

Chapter 2

Identity and Access Management

Identity management (IDM) is about managing multiple user's sensitive information on a platform that enables smooth and secure transactions. This used to be a job that was handled manually in the past, but the constant growth of technology and the use of it in commercial markets, lead to a need for automatic identity management. Identity management is often coupled with access management (AM) because they support each other, this combination is referred to as Identity and Access Management (IAM). IAM is closely linked to security and ethicality because there are real identities of people involved.

Identity management is all about handling sensitive user data. Usually, the case is that a person joins a company and has their credentials created into the company database. Then the user has to be assigned some role and other accounts added to them like word processing tools e.g. this process of adding the information and storing automatically is what IDM is mean to do. This can be done manually in smaller size companies, but when there are thousands or hundreds of thousands of users the automatic provisioning of users and their rights is necessary. Enterprises have been adopting cloud-based solutions and almost all take advantage of cloud computing in some way [5]. The use of cloud applications means that users will have multiple accounts and they all need to be handled in some way.

2.1 Components of Identity and Access Management

Identity and access management strives to solve the problem of how to handle large amounts of users in a system. It can seem like a nontrivial task, but in reality, there are multiple challenges when handling people's data. Security is the biggest challenge in IAM and should be implemented from the beginning. A working identity management service must have a way to store and identify users and groups of users. Updating, deleting, and modifying the users and groups is a necessity for the system to be considered an IAM service. When handling sensitive information it is not good practice to allow access for everyone to the data, that is why an IDM service needs role control. Most of all the system and it's sensitive user information must be protected from malicious activity. [6, 7]

IAM service should be a tool that makes understanding, controlling, and maintaining the underlying databases and user management clear and simple. This means that admins should be able to view and modify user rights using a user interface. Identity management is part of IAM, which contains the management of user and group data. The main task of IDM is to create automated workflows of user creation and updates, this automation will reduce the risk of wrong information flow. The use of automatic workflows speeds up the process of provisioning users and groups besides it also helps with creating essential user accounts. Meaning that the IDM service will automatically create the new user the desired accounts e.g. Microsoft office account. Even when the system is automatic to a certain degree it still needs to give administrators the tools to modify the access rights and other information of a user. [6]

Access management is meant to enforce security policies and grant access to users that are allowed to access that data. Access management also monitors the network usage onto the service so that it can be analyzed if needed. Access management is the driving force of security and authentication. Authentication is used to identify the user's access rights to the services provided by the client. This means that a user who has administrative rights, for example, will see a different view than a user with normal access.

Identity and access management can work separately, but to ensure that the service is secure the two should be used together. The authentication and authorization will be done with access management and when the user is authenticated then identity management can start working. This kind of relationship is what most IAM systems have.

2.2 IAM Lifecycle

This section will discuss the lifecycle of identity in the IAM system. When an IAM system is being set up, it needs to be provisioned from somewhere. Provisioning can happen from an existing database, document, or be manually added. This initial phase of the lifecycle is where all users get their access roles based on their roles and other necessary information. This initial provisioning can take a long time since every user and group needs to pass the workflows built in the IAM. Authentication and authorization is something that is the next step of the lifecycle. This part is important to be built correctly to meet the needs of the client because authentication and authorization is the part that enforces security policies. These first two steps can be considered the buildup phase where the service is being built and configured to meet the requirements of the client. [8, 9]

Once the service is set up, it needs to be maintained to have accurate and up to date information. Maintaining happens with administrators having control of the IAM and users having access to their data to update them. This can be achieved because the authorization and authentication part has been implemented so that users and admins can be recognized. During this phase, IAM can be used to its full capacity and it opens up Single Sign-on (SSO) for use. In today's time, users have accounts on several cloud-based services and SSO enables users to sign in only in one place and have access to all the integrated services. [8, 9]

The last phase of the lifecycle is Auditing, this means that everything in the system should be saved and be viewable to the client. The recording of data makes it easier for

the client to see misuse and possible security breaches. The handling of this material produced by IAM could be used by another service that is built to maintain security in the client's system. The IAM lifecycle is not necessary a lifecycle with a definitive start and end, because the system will run through these processes when necessary. For example, the provisioning should be run every time a user leaves or joins the company. The next sections will discuss these phases of IAM in more detail explaining use cases and used standards. [8, 9]

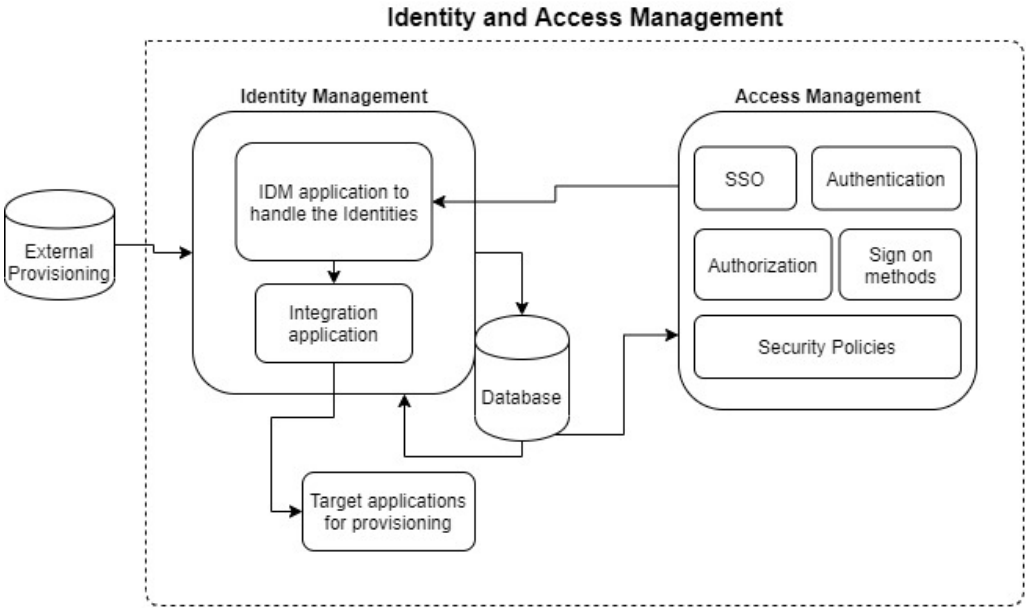


Figure 2.1: Example of an IAM model

2.2.1 Provisioning

Provisioning means the process of maintaining all entities in the IAM system. Figure 2.1 illustrates a simple IAM model where the identity management and authentication management are separate; this figure will be a reference point while examples are discussed. Provisioning is the first thing that happens when a new IAM service is put together because otherwise, the whole system would not have any users to work with. In terms of how the initial provisioning will happen is up to the client that buys the service. Usually, the

client already has some kind of list or database containing the identities and access rights. The problem to solve is how the identities are provisioned into the system. IAM systems should not be customized for all clients and that is why the IAM systems should accept data from commonly used HR applications, databases, and files like comma-separated value files. The initial provisioning will make sure that the IAM database is the master database that is going to be used. [8, 9]

After the initial provisioning, the IAM system is going to act, as the truth, meaning provisioning and access management will be done based on the data found in the IAM database. Additional external provisioning is still possible but only to add new users otherwise, the data should be updated in the system. This kind of process has the benefit of making the system work faster, but on the downside, if the client does not handle provisioning automatically when the user leaves the company, it creates a security risk. For this reason, the IAM system should have automated scheduled provisioning. The schedule of this automatic provisioning is based on the client's need e.g. if the client has one person change every year the schedule can be monthly but in larger companies, the schedule can be once per week or more. The concept of this scheduled provisioning is to keep the system up to date and have the users the needed access to external applications. [9]

Provisioning also handles the account's rights to external applications that the client uses. This is handled in the identity management section in provisioning as figure 2.1 shows. The idea is to have the IAM system act as an adapter and send the required data to the trusted external application. This type of provisioning allows the system to be very effective and save time compared to creating the accounts separately. There is also the addition that when an identity is deleted or removed from a group, it will modify the accounts access to the external applications as well. Provisioning is important for the whole system to work and should be done correctly, otherwise, the system will not operate properly. [10, 11]

2.2.2 Authentication and Authorization

Authentication and authorization are responsible for security in the system. It is divided into two different categories. First being Authentication that is responsible for checking that the user is the correct identity. This can be seen as using a password and username to authenticate the user. Authentication also handles session management, so that users can't be signed in for longer than one session. Authentication provides SSO with keeping the session data in cached memory and then deleting the session after the user leaves. Authorization is there to keep control of what users can do and where they have access. The most common access control is to have role-based control. There can also be rule- and attribute-based controls. The role-based access control means that only users with a specific role can access for example documents or web pages. These roles are given to users during the provisioning state. To better understand authentication and authorization we will look at the history of it. [12]

The evolution of access management can be divided into three generations. The basis of access control lies in having a directory that contains information about the entities inside the database. The first generation of access control used this basic idea for their protocols, such as X.509, Kerberos, LDAP, and Radius. All of these technologies have been developed in the '90s and they have set the foundation of access control. Lightweight Directory Access Protocol (LDAP) uses a network of directories that all have the same data in them. This allows the LDAP to verify the given username and password by querying the directories for the known data. The query result will determine if the client gets authenticated. There are two different authentication methods used in LDAP. The first method is the basic authentication where the client gives a password and username. The second one is administrative authentication where the client can read and write the information in the directory. [11, 13]

Kerberos is another security protocol that can be used to authenticate the client. It was developed at MIT in the 1990s to authenticate in open networks. Open networks mean

that a client should be able to authenticate itself without sending password information in the message. All the messages in Kerberos are encrypted so that it is harder to read the authentication information. The idea of Kerberos authentication is to have the client send a request to an authentication server to acquire an encryption key. With this encryption key, the client can verify itself with the target application. The encryption key that is acquired from the authentication server is a session key that has a specified time it will work for. When the time ends the client needs to send a new request to the server. The use of session keys prevents using old encryption keys for malicious intention. [11, 14]

X.509 is a standard that uses a public key infrastructure (PKI), which is used in multiple different internet protocols such as the transport layer protocol. The idea is to use public key infrastructure to authenticate the identity. PKI uses a pair of keys where the other key is private and the other is public. This will mean the client and user will have their private keys and one public key that they will use as authentication. X.509 certificates are used to store these pieces of information and they have a validity period so that they can't be used forever. The maximum validity period of the x.509 has been capped at two (2) years starting from 2018 [15]. [11, 16]

The last authentication protocol is Remote Authentication Dial-In User Service in short Radius, which is used for remote authentication. Radius uses datagram messages that are sent to a specific port on the client machine. Radius provides both authentication and authorization, it also has an additional component called accounting. Accounting means that the service keeps track of data that has been transferred in the session. This is especially handy for internet service providers for tracking the amount of data going through their network. [11]

The second generation of access control technology was developed to meet the requirement of users having multiple accounts that need to be authenticated. The solution to the problem was to use open standards so that service providers can accept messages from clients. The solution is called cross-domain identity federation. Cross-domain means that

a client can have multiple domains that it uses with one authentication. The federated part means that someone or something is handling the authentication and rights of the client. Cross-domain also opens up Single Sign-on and account provisioning that was mentioned in figure 2.1 of an identity and access management system. Cross-domain needs some kind of authentication for it to work, here the use of security tokens is very common. Security Assertion Mark-up Language (SAML) is a security token that has been commonly used for authentication. SAML is a good choice for the security tokens because it does not need an application-specific credential for it to work, which is a necessity when implementing SSO. [11, 17]

SAML works the same way as many other protocols, first, the user sends a request for a SAML token to an identity provider and then the identity provider gives a token back if the credentials are correct. After this, the user can make calls to the service provider with the token it has received. Service providers and identity providers need to form a trusted connection to create valid security tokens. SAML has been updated to version 2.0 and it is still considered a good option and is still used widely in access management systems. [11, 12, 17]

Third-generation access control technologies are still being developed and adopted into new systems. They share the basic principals of second-generation technologies but have made improvements to the protocols. Oauth and OpenID are good examples of these third-generation authentication and authorization methods. Oauth is not an authorization method meaning that it's supposed to work with another protocol for authentication. To solve this problem OpenID Connect was made so that there is a full package. This implementation combines the authorization of Oauth 2.0 and authentication of OpenID into one bundle making up a full access control protocol. [11, 18]

OpenID Connect works with API calls that are done to an OpenID provider. Application program interface (API) is a communication protocol that is used to retrieve data from a source. HTTP calls are API calls that are sent to the server and the server then

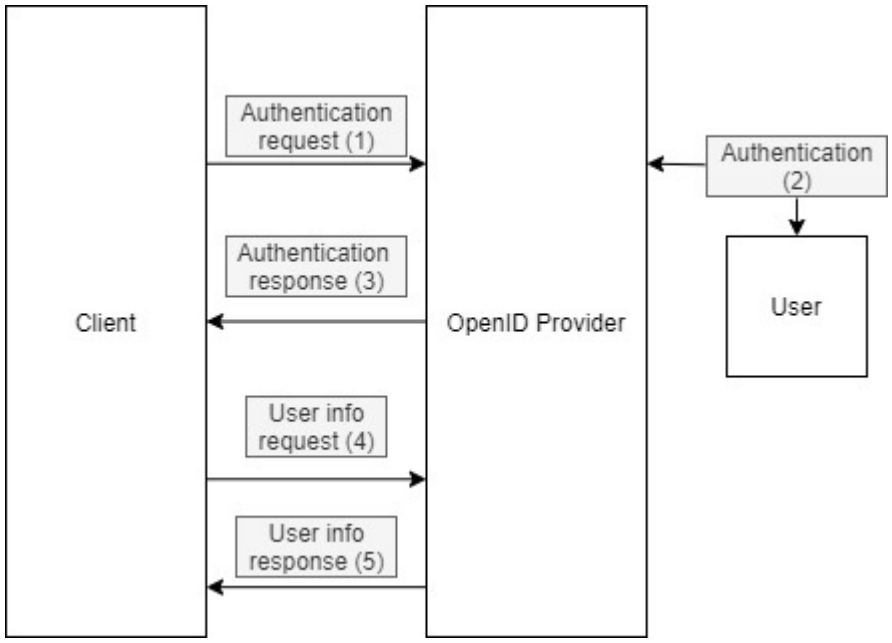


Figure 2.2: OpenID Connect concept

responds with the data that was requested. In OpenID connect these API calls are made from the client to the OpenID Provider. The Client can be anything from a web application to a mobile device application, it does not matter just that it understands the responses from the OpenID provider. Figure 2.2 illustrates the basic concept of authentication and authorization in OpenID Connect. First, the client sends an authentication request to the OpenID provider. After this, the OpenID provider authenticates the user and responds to the client. When authentication is done, the client can send a user info request where the OpenID provider will respond with the information. The evolution of the protocols in authentication and authorization has been made in the encryption and information format. The basic concept of authentication has stayed the same because there is no need to reinvent the wheel. Most important authentication and authorization are all about protecting sensitive data. [18]

2.2.3 End of IAM Lifecycle

The rest of the Lifecycle that IAM holds is more implementation-oriented rather than protocol-specific and that is why the last three steps will be discussed together. The third step was self-service and that means the IAM system can handle requests from users to change passwords and their data. This step could not be achieved without the first two steps and is where the users see the IAM at work. In a sense, this step is where the minimum requirements of an IAM system lie. It would not be practical to leave the system into this state because there would be no use to it. That is why the last two steps of the lifecycle are needed. Password management is the fourth step and the meaning of this step is to have federated systems in the IAM system to enable SSO on cloud-based services. Password management is also responsible for how the passwords are stored in the cloud-based database. Passwords must always be encrypted to ensure that no one can get them easily. Also, it should be considered that the passwords should not be transported if not necessary. [11, 9]

The last step of the IAM lifecycle is compliance and audit. This step ensures that everything done in the system should be tracked and those logs are kept safe until deemed outdated. This will help the system be more secure and in the case of a security breach, the attacks can be noticed. Auditing also helps the auditors to gather reports and check if access control policies are enforced. These five lifecycle steps are what make up the lifecycle of an identity. In a sense, these steps are not supposed to be parallel with a timeline, but rather the steps that identities will take in an IAM system. As an example, an identity will start with the provisioning and when the identity leaves the system it will go back to step one also known as deprovisioning. To conclude the Lifecycle of an IAM system is mostly a visual way to see what should happen in the system. It is not a definitive guideline nor protocol that should be followed. [11, 9]

2.3 Identity Management as a Service

In recent years, everything is being taken into the cloud, this trend is also visible in identity and access management. What this means is that companies can outsource their identity and access management to another company providing identity as a service (IDaaS). Changing from a regular company-wide application to a cloud-based service requires that the vendor has a fully built and automated service ready. The use of cloud-based solutions enables vendors to have multiple tenants running on the same identity management system. The system tracks the access of tenants with access control so that private data is not mixed. With having multiple tenants running on the same cloud-based system, security is the most important. [9]

All IDaaS vendors provide certain basic services in their products. These basic services are authentication services, account management policies, and federated access control. Authentication services work with the federated access control to enable SSO to all the other cloud-based applications. Account management policies are for the administrators of the system to limit the access of regular users in the system. User provisioning is something that belongs to the client's responsibilities. Clients also need administrators to handle the data in the service and audit the system. In Microsoft Azure AD user provisioning can be handled from an outside service. This means that if the client has already existing identity management software, but wants to implement access control they can just add their software to automatically provision to Azure AD. This brings us to the main focal point of this thesis, a system for cross-domain identity management (SCIM). Azure AD uses SCIM messages to provision and deprovisioning, this helps the systems to update data and keep it the same for both parties. [9, 10]

2.4 System for Cross-Domain Identity Management

System for Cross-Domain Identity Management (SCIM) is a specification for managing identities in cloud-based applications. The problem SCIM aims to solve is simplifying the ways of handling and storing identity data across applications. The priority in the SCIM specifications is on simplicity and creating a standard way to exchange user information. SCIM uses already existing authentication and authorization methods for access control and it does not specify how they are to be used, what SCIM does is create a specification that works well with the already existing privacy models. SCIM tries to create a simple and easy to use specification that makes identity management cheaper to handle in the cloud. SCIM is an application-level protocol that helps with managing identities and provisioning them. The protocol uses HTTP methods for data exchange e.g. GET as a way to retrieve identities. [3, 2]

The first version of SCIM was released in 2011 and it was named Simple Cloud Identity Management. The core schema provided a protocol for representation in both JavaScript Object Notation (JSON) and Extensible Markup Language (XML). During that time, SAML was used for access control, and thus the documents covered how to map SCIM attributes to SAML for enabling just-in-time provisioning for SSO. The second version called SCIM 1.1 was released in July 2012 changing the name to System for Cross-Domain Identity Management. It was not a major release, but rather a cleanup for the issues that had been found in working cases. Only the core schema and Representational State Transfer (REST) API calls had changes made to them. SCIM 2.0 was released in September 2015 and it is still the version that is being used for all new SCIM adaptations. Before going deeper into what the SCIM specification holds in it we should understand what cross-domain identity management is. [1]

Cross-domain identity management is what SCIM tries to solve. Identity management has been explained in the previous chapter. Cross-domain means that the whole system should be able to work on multiple different platforms. Figure 2.3 shows the basic con-

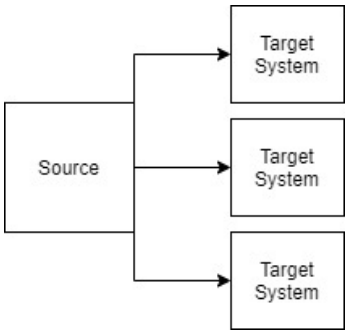


Figure 2.3: Cross-domain concept

cept of what a cross-domain system should do. The source system will be connected to multiple different target systems. This allows the source to work as a single point of contact allowing cross-domain operations to happen. In identity management, this means that the source system will have up to date information about users and it will use that data to provision the target systems. This allows the source to create accounts for one user on all or selected target systems and enable SSO for the users. SCIM provides the tools to accomplish this kind of functionalities. SCIM should not be thought of as a separate system, but rather an adapter for already existing applications to talk the same language.

This chapter covered the basics of what the IAM system is and what kind of components it has. The main point was to have a clear view of what identity management and access management mean and what they are used for. The thesis will examine a commonly used cross-domain identity management specification called System for Cross-Domain Identity Management, the next chapters will cover the core schema and protocol of this system.

Chapter 3

SCIM 2.0 Core Schema

From now on whenever, SCIM is mentioned the reader can expect it to mean SCIM version 2.0 published in September 2015, unless otherwise stated. SCIM specifications have been split into three different Internet Engineering Task Force (IETF) Request for Comments (RFC) documents. One of these documents is the core schema of the SCIM specification, defining how the model for the information should be built and stored in a database.[2]

The core schema document introduces the schema of attributes that are used in building the identities. It also describes the resources that can be made from these attributes. The idea is to have set attributes that either must have a value or not. From these values, the identity, which is called a resource in SCIM, is created. The Model of SCIM is a tree structure where the resource is the root. This kind of root-based model helps in delivering a standard model that can be modified by inheritance. Next sections will discuss the SCIM schema and model.[2]

The definitions of the core schema are listed below for future reference:

Service Provider

An application that uses the SCIM protocol to handle identities using an HTTP API.

Client

The application that uses SCIM protocol and HTTP request to receive and manage identities in the Service Provider.

Resource

A root object that contains attributes and is managed by the service provider.

Resource Type

The type of resource that indicates what the resource holds inside. This can be for example user or group.

Endpoint

The endpoint is the URI that indicates where to call the API to receive certain resources e.g. the URI to get user information is "https://baseurl.com/v2/users" where the endpoint is users. This base URL is the service provider URI.

Schema

A set of rules that define what attributes can be e.g an attribute can be of type = string, so it does not accept numbers as an input.

Singular Attribute

An attribute that only has a single value e.g. ID.

Multi-valued Attribute

An attribute that can contain multiple values such as phone numbers

Simple Attribute

An attribute that can be singular or multivalued, but cannot contain sub-attributes also the value must be primitive.

Complex Attribute

An attribute that can be singular or multi-valued and contains single or more simple attributes.

Sub-Attribute

A simple attribute that is tied to a complex attribute.

3.1 SCIM Schema

SCIM schema is the backbone of the whole specifications because without the attributes that make up the schema it would be impossible to invoke rules on a model. SCIM schema has not been made up just from scratch, but rather it gathers information and best practice knowledge from previous identity style protocols and schemas. For this reason, it can be said that the SCIM schema has a good background from the fact that it uses working parts from each model to create one simple version. [2]

The SCIM schema is made up of attributes that have certain values, they can be thought of as building blocks for the resource to be built. SCIM schemas are built to tell the service provider what to do with new information on a specific attribute, the schemas can be thought of as guidelines on what an attribute can be. The SCIM document calls these guidelines, attribute characteristics. All the schemas in SCIM documentation are in JSON format and use camelCase for the naming convention. Camel casing means the words are written with no spaces and each new word starts with a capital letter except the first word. Listing 3.1 gives a overview of attribute characteristics used in SCIM. [2]

3.1.1 Attribute Characteristics

The first characteristic is the name of the attribute, which is quite straight forward. Type is the attributes data type and it can range from primitive types e.g string, integer, boolean to dateTime, and reference. The last type is complex and this means the attribute is a complex attribute where the subAttributes are mentioned. This can be seen in the listing 3.1 on Name and Emails characteristics, the sub-attributes have all the same characteristics as the original attributes, meaning sub-attributes can have sub-attributes in them. The next attribute characteristic is multiValued which defines if the attribute is singular or multi-valued with a boolean indicator, where false is singular and true multi-valued. [2]

A description is a readable definition of what the attribute is as can be seen in the listing 3.1 line 5. Required and caseExact are both boolean values where required means if the attribute can be empty or not. CaseExact means if the attribute value is case sensitive, meaning that if the word can be processed as all lowercase or must the cases be used in processing e.g. when matching names. CanonicalValues are are suggestions of values the attribute can have, which can be seen in line 45 where the type of emails can be home, work or other. [2]

Mutability is the characteristic that defines how the attribute can be modified. There are four different values for mutability. ReadOnly means the attribute can only be read not modified. ReadWrite means the value can be changed. Immutable means the attribute can only be created when the resource is created. WriteOnly means the value will not be returned but can be modified at any time. [2]

Returned dictates whether the attribute can be returned to a client that makes a request. The values used are: always, never, default, and request. Always mean the attribute is returned whenever a request is made, never means the attribute is never returned no matter what. Default means the attribute is returned by default in all requests where attributes are returned, but if a GET attributes request is sent the value will only be returned if stated in the request. The request means the attribute is returned in every POST, PUT and PATCH where the attribute is modified. [2]

Uniqueness is the last common attribute characteristic and it has values of none, server, and global. None is the default value and it means that the attributes are not unique in any way. Server means that the value should be unique in the SCIM endpoint or unique to the tenant at hand. Global means the value must not be the same in the whole service including different tenants. These attribute characteristics build the schemas that are necessary for the service provider to know how the attributes should be handled when a client send request. [2]

Listing 3.1: SCIM Schema presentation

```
{
  "name": "userName",
  "type": "string",
  "multiValued": false,
  "description": "Unique identifier for the User.",
  "required": true,
  "caseExact": false,
  "mutability": "readWrite",
  "returned": "default",
  "uniqueness": "server"
},
{
  "name": "name",
  "type": "complex",
  "multiValued": false,
  "description": "The components of the user name",
  "required": false,
  "subAttributes": [
    {
      "name": "familyName",
      "type": "string",
      "multiValued": false,
      "description": "The family name of the User.",
      "required": false,
      "caseExact": false,
      "mutability": "readWrite",
```

```
27            "returned": "default",
28            "uniqueness": "none"
29          }
30          // ... code removed for readability
31      },
32      {
33        "name": "emails",
34        "type": "complex",
35        "multiValued": true,
36        "description": "Email addresses for the user.",
37        "required": false,
38        "subAttributes": [
39          {
40            "name": "type",
41            "type": "string",
42            "multiValued": false,
43            "required": false,
44            "caseExact": false,
45            "canonicalValues": [
46              "work",
47              "home",
48              "other"
49            ]
50          }
51          // ... code removed for readability
52      }
```

3.2 SCIM Model

The SCIM protocol uses a tree model to define what attributes there need to be in a resource. This kind of model means that the resources have a child-parent relationship. Inheritance means that all the resources have some kind of common attributes that they all share. Figure 3.1 illustrates the ready defined resources from the SCIM protocol. The protocol itself does not regulate the extension that can be made to the resources, rather it dictates that they must inherit all the attributes from their parent resource. The root node is called Resource and holds inside the common attributes that are used in all other resources. [2]

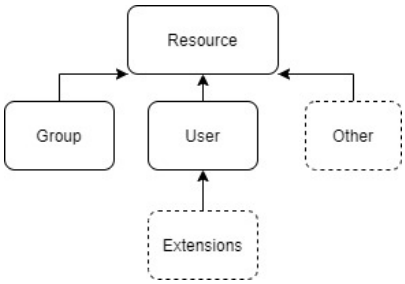


Figure 3.1: SCIM Model

The common attributes are as the name states something that all resources must include in SCIM. They are id, external id, and Meta, which is a complex attribute. The id is a unique identifier that is given by the service provider to the resource to track the specific resource. The id cannot be empty or null and it has to be unique so that there are no two id's withing one service provider. The external id is an identifier that indicates the id of the resource in a client system. This value is assigned by the client and not the service provider, and it can be null if the client does not specify anything. [2]

The last common attribute is meta, and inside the meta attribute, there are the sub-attributes called, resourceType, created, lastModified, location, and version. This metadata is maintained by the service provider for all resources. The resourceType sub-attribute specifies what the resource is e.g. a user type. Created and lastModified are time identifiers and are exactly what the names state. The last-modified attribute will be updated every time the resource is updated. The location attribute is a URI containing the location of the resource using the id that the resource has e.g. baseurl/v2/users/"id". The version attribute tells what the version of the resource is and it can be used to validate that

the latest version of the resource is used. One last attribute that must be included in all the resources is the Schema attribute. It shows the schemas that should be used to handle the data that the resource owns. [2]

3.3 Users

SCIM works with two preset resource types one of them being a user type. Figure 3.1 shows the inheritance structure used in SCIM. The user resource type inherits the common attributes from the root resource, meaning the user must have all the common attributes inside it. To make the resource more useful there are additional attributes that make up a user identity using the SCIM standard. These attributes are divided into singular and multi-valued attributes as described in the schema section (see section 3.1.1). These additional attributes help build a database with information that can identify a user. The user schema URI is "urn:ietf:params:scim:schemas:core:2.0:User". The singular attributes that are included in the SCIM schema are listed below.

3.3.1 Singular Attributes

userName

Username is a string format name that is unique to the service provider. This attribute is used to authentication by the user to the service provider. This can be seen as a more user-friendly view compared to the id. This attribute is required and cannot be null.

displayName

The displayName is a representation that is viewable to others and the user. It should be the full name of the user, but it can also be a handle that is used. For example "Mr. John Doe II", but if needed the format can be "jdoe". It is up to the service provider to decide the format.

name

The name attribute is a complex type and it contains sub-attributes that are formatted, familyName, givenName, middleName, honorificPrefix, and honorificSuffix. The family-, given- and middleName are the names of the person as described. The honorificPrefix are titles that a person might have e.g Mr, Mrs, Dr, etc and the suffix is an addition like "III". The formated sub-attribute is the whole name displayed in the order that is prefix - givenName - middleName - familyName - suffix.

nickName

The nickname of the user that is used in real life. This attribute cannot have the same value as the userName.

profileUrl

This is the URI that the user resource can be found when calling a GET request.

title

The title of the user mostly used for work-related titles since the honorificSuffix contains titles attached to names.

userType

The user type shows the relationship of the user with the company that has stored the identity. This is another work-related attribute.

preferredLanguage

This attribute shows the language the user prefers to speak and read if possible. It can be used by client-side applications to determine the default language to show. This attribute is formated the same as the HTTP Accept-Language header field RFC 7321. [19]

Locale

Defines the location of the user for addressing information that is unique to the location e.g. currency and date-time format. This is again used by a client-side application.

timeZone

This attribute represents the timezone of the user and is formatted in the IANA Time Zone database format [20].

active

This attribute is a boolean value that can be either true or false. The meaning of this attribute is decided by the service provider, the most common case is to show whether the resource is usable or not.

password

This attribute is the password used by the service provider to authenticate the resource. The password is not returned in any format to the client by the service provider. The password can be changed by the client and the service provider stores the encrypted password. SCIM does not make standards for password management, but rather gives the service provider the ability to use the most up to date security method.

These single attributes make up one portion of the user resource together with the common attributes that are inherited from the root resource, also, there are multi-valued attributes that are defined below.

3.3.2 Multi-Valued Attributes

Most of the multi-valued attributes have the same sub-attributes. Unless otherwise stated the sub-attributes used are value, primary, and type. The value represents the string value of the attribute that is visible to users. Type represents what kind of value the resource is e.g. work or home in most cases. Primary shows if the attribute value is used as a primary source of information that is shown to end-users.

emails

Emails attribute contains the emails a user has and the primary email will have a boolean value of true in the primary sub-attribute. The email value will be a cleartext format of

the whole email e.g. test@testnetwork.com. Emails attribute uses the standard sub-attribute set as mentioned above.

phoneNumbers

Phone numbers represent all the numbers associated with the user and includes the primary choice of the user.

ims

Ims represents all the instant messaging applications that the user has access to, there are no preset values that are required to have so the service provider can accept a string value for the name of the ims.

photos

Photos contain all the photos of the user. These photos are not photos taken by the user, but an image of the user to use as a thumbnail in applications. The value sub-attribute of this attribute is a URI where the image is located. The location should not be a web page, but rather an image file.

adresses

Addresses is the first attribute that has a different sub-attribute format. Here the sub-attributes are formatted, streedAddress, locality, region, postalCode, and country. Street address is the physical address of the street. Locality means the city or something comparable, region is the state or something comparable to it. Postal code is the number used by the country for postal codes. Country is the country that the user has the address in, the country should use the ISO 3166-1 alpha-2 code format, which is a two-letter code for each country. The formatted sub-attribute is the combination of all the other sub-attributes to make a human-readable address.

groups

Groups is an attribute that lists all the groups the user is in. The sub-attributes are display, value, and $ref. The display sub-attribute contains the name of the group and the value holds the id of the group the user is in. $ref is the URI of the group resource.

entitlements

Entitlements are a list of achievements or rights the user has. These entitlements can be used to allow special access to certain files or services.

roles

Roles are something that describes how the user is related to the client. The value can be anything as long as it is in string format. The value can be e.g. "employee" or "owner". These are not the same as title.

x509Certificates

The x509Certificates is a list of certifications that belong to the user. Each value that the certificates contain is one X.509 certificate that is base 64 encoded.

The user resource contains attributes that are deemed as necessary information when creating an identity. The reason for having work-related attributes in the user resource is because SCIM is aimed at corporate and governmental use. The user resource does have an extension called enterprise user for additional attributes required in enterprises. The mix of regular identity data and the work-related data such as roles in the user resource make it a good base for building an identity. The simplicity of the data is very important because it helps client applications to create users based on the provisioning data sent by the service providers. [2]

Thinking of the user resource in standard use the only attributes that could be removed are title, userType, and roles. This is to say that without these three attributes the identity would still be usable. This means that not only does the SCIM user resource work on a corporate level, but it also works in a more relaxed environment. The biggest advantage is that most of the attributes are not required and can be used if needed. This flexibility is what makes SCIM so usable and good.

3.4 Groups

Groups is another set resource that is defined in the SCIM core schema. Groups and users are the two main resources that are used in SCIM implementations and they are the basis of actual programs that use SCIM. This section will focus on what attributes the group resource has. The SCIM core schema defines that the group resource must have a schema URI that defines the resource as being a group resource. This URI is "urn:ietf:params:scim:schemas:core:2.0:Group", the value is located in the common attribute schemas. Below is a list of the attributes that the group resource has.

displayName

The display name for a group will be stored in this attribute and it is the human-readable version of the group. This value does not need to be unique to the service provider.

members

This attribute is a complex type and it is a list of users that are in the group. The sub-attributes included in the members attribute are value, $ref, and display. The value attribute is the id of the member. $ref is the URI of the resource which means where to find the actual resource. Display is the human-readable name of the resource in the members list for a user the display would be the name of the user. There is a possibility in nested groups which means that one group can have multiple groups in its members list. In this case, the sub-attributes will have the nested group information.

The group resource does not have many attributes, but it contains all the necessary information that is needed to group people. The members $ref and value sub-attributes are immutable meaning that the values stored cannot be changed unless the resources are deleted from the group. This helps with keeping the data intact in case of mistakes. The good thing is that display can change in the members list when a change is made to the resource.

3.5 Enterprise User

The enterprise user is a resource that inherits the user resource and is thus considered an extension model. This resource is the only defined extension in the SCIM core schema. The extension was made to include data that is considered commonly used in enterprises and helps companies to integrate SCIM into their systems. The enterprise schema URI is "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User". The attributes that are included in the enterprise user are:

employeeNumber

The string format of the user's employee number. Generally can contain numbers and characters.

costCenter

A representation of the users cost center

organization

A representation of the user's organization

division

A representation of the user's division

department

A representation of the user's department

manager

A complex attribute that contains information about the user's manager. The complex type allows the service provider to show the hierarchical structure of the company. The sub-attributes are value, $ref, and displayName where value is the if of the manager and $ref the location where to access the resource. The display name is an optional value and it shows a human-readable value of the manager's name.

The enterprise user is a good example of what additions it's possible to make for certain cases when using SCIM. The fact that SCIM enables extensions in its specification

helps fight the test of time. These extensions will fill in the holes of missing information if clients want to customize the product for them.

3.6 Service Provider Configuration Schema

The service provider configuration schema defines how to represent the configuration of the service provider. This is used to show what operations and schemas the service provider is capable of accepting. This helps the automation of the services when the client can determine the operations in use. The configuration schema is the only schema where there is no id required. The schema URI is "urn:ietf:params:scim:schemas:core:2.0:ServiceProviderConfig". The attributes defined in this schema are:

documentationUri

The location of the human-readable help documentation that can be accessed through an HTTP address. This is optional for the service provider.

patch

A complex attribute where the sub-attribute is supported. Supported is a boolean value and it defines if the service provider accepts a patch request.

bulk

A complex attribute that defines if the service provider accepts bulk requests. The sub-attributes are: supported, maxOperations, and maxPayloadSize. The supported sub-attribute is a boolean value and determines if the operation is supported or not. Max operations define the value for the maximum amount of HTTP operations accepted in a single batch request. Max payload size is the byte value of the maximum allowed size of the batch file sent.

filter

A complex attribute that defines if the service provider accepts filter requests. The sub-attributes are supported and maxResults. The supported sub-attribute defines

whether filtering is available with a boolean value. MaxResults is an integer value of the maximum amount of results returned during one filter request.

changePassword

A complex attribute that defines if the service provider supports password management. The sub-attribute is supported and it is a boolean value that defines if the operation is supported.

sort

A complex attribute with one sub-attribute that is supported. The sub-attribute is a boolean and defines if the sort operation is supported.

etag

A complex attribute with one sub-attribute that is supported. The sub-attribute is a boolean and defines if the ETag operation is supported.

authenticationSchemes

Authentication schemes is a multi-valued complex attribute that shows the available authentication methods accepted by the service provider. The sub-attributes are type, name, description, specUri, and documentationUri. The type specifies what the authentication schema is e.g. "oauth2". The name is the commonly used name for the authentication schema. The description contains a description of the authentication schema. Space URI is an URL point where to find the authentications schema. Documentation URI is a URL where to find the usage documentation of the authentication.

3.7 ResourceType Schema

The resource type schema determines what kind of resource types the service provider uses. This is extremely useful for clients to understand what resource types they need to use to exchange information properly. The schema URI is "urn:ietf:params:scim:schemas:core:2.0:ResourceType". The attributes are:

id

The resource types id on the server. This is an optional attribute because the name attribute can be used for this.

name

The name attribute defines the name of the resource type, when possible the name must be specified.

description

The description attribute describes the resource type.

endpoint

The endpoint attribute defines the URL where the resource type can be accessed e.g. for users the endpoint is "baserl.com/v2/users".

schema

The schema attribute contains the URI of the resource type. These URI are shown in the sections above.

schemaExtensions

The schema extensions attribute is a complex type, it shows the extensions that the shown schema may have e.g. the user schema has an extension of enterprise user. The sub-attributes are schema and required. The schema sub-attribute shows the URI for the extension. Required is a boolean value that indicates whether the extension is required of not.

3.8 Schemas

The schemas are defined in the SCIM core schema documentation and they work as guidelines on how each attribute is handled. These have already been shown in chapter 3.1. The idea is that each resource has its schema that defines how the attributes are handled, the JSON code 3.1 is a good example. The URI for the schema is "urn:ietf:params:scim:schemas:core:2.0:Schema". The Schema attributes are:

id

The id in schemas is the URI of the resource that it is tied to. This means that if the schema is the schema of users then the id is "urn:ietf:params:scim:schemas:core:2.0:User". This makes it easier to combine the schema and resource type.

name

The name attribute is the human-readable name of the schema. For example, if the schema is for the group resource the name will be Group. This attribute is optional.

attributes

The attributes attribute is a complex type that defines how the service provider configures the attributes. There are multiple sub-attributes and they are:
name, type, subAttributes, multiValued, description, required, canonicalValue, caseExact, mutability, returned, uniqueness, and referenceTypes. The definitions of these attributes can be found in chapter 3.1

3.9 Summary of the SCIM Core Schema

The SCIM core schema document lays the foundation for the whole protocol. It shows the responsibilities of the service provider and the client, but it does not discuss how the actual communication will happen. The document defines the regularly used resources such as user, group, and enterprise user. These three resources are what make the SCIM application usable to most clients. The other resources such as the service provider configuration schema and resource type schema are made to support the SCIM applications. The schema resource was made so that the other resources have a way to configure the attributes modification and returnability on the service provider side.

The most important thing the core schema gives is the model on how to build the resources. These attribute definitions help understand how the resources are tied together. The flexibility of the attributes and extension makes it possible for the SCIM specification to work in multiple different situations. The schema is well written and provides the necessary building blocks to create a customizable service for identity handling. The next thing in the SCIM specification is the protocol, that describes how the service provider and client send data between each other, this will be the topic for the next section.

Chapter 4

SCIM Protocol

The SCIM protocol determines how the HTTP-based protocol will exchange data between the service provider and the client. This documentation leans on the SCIM core schema [2] for the resources that it uses. The protocol aims to give a regulated way of exchanging data within the SCIM domain. The document does not give any example code nor does it restrict the methods of creating the actual product. "The SCIM protocol is an application-level HTTP-based protocol for provisioning and managing identity data on the web and in cross-domain environments such as enterprise-to-cloud service providers or inter-cloud scenarios." [3]. The protocol defines how the creation, modification, and retrieval of identities happens. The protocol also states how the resources, defined in the core schema, can be accessed. This chapter will use the SCIM protocol documentation as a reference if not stated otherwise. [3]

SCIM is based on HTTP and it uses the standard format of headers and URIs defined in the HTTP documentation [19]. SCIM also takes advantage of JSON format which means the payload is formatted in JSON when sending data. The JSON format that is used is defined in the HTTP header field as "application/scim+json" so that the receiving side can determine this message is using SCIM format in the message. Using the JSON format in the resources creates a good base on how the messages can be interpreted by the receiving party.

4.1 Authentication and Authorization

Authentication is something that should always be thought of when handling data that is sensitive. Therefore the SCIM protocol does acknowledge that authentication and authorization must be used, although it does not limit what methods are used. The SCIM protocol requires the use of Transport Layer Security and standard HTTP authentication and authorization schemas. The schemas in section 2.2.2 shows the three generations of HTTP authentication because the protocol does not limit the choice of authentication and authorization it would be best to use the most up to date methods. In this case, the choice would be to use Oauth2.0 and OpenID or just OpenID Connect, which leverages the Oauth2.0.

Although the protocol does not restrict the methods used in security it does require that the service provider must have the ability to map the authenticated client to an access control policy determined by the service provider. Putting it in more common language means that an authenticated client, in this case, let it be a user, must be found in the service providers database and have the rights to access the information that is requested. This means that authentication does not mean full access to everything. The SCIM protocol does take into consideration the security and everything necessary in identity management, but it does not enforce any rules that would limit the use of this protocol. This is one of the good things about SCIM that it aims to be a backbone that can be built upon when new methods are invented.

4.2 Endpoints

The endpoints defined in SCIM are the primary resources from the core schema, this means the user and group resources have their endpoints that can be accessed with HTTP calls. These endpoints are enforced by the service provider and must conform to the standard set by the SCIM protocol. The service provider can have more endpoints in use

depending on the extensions it uses. The client can find all the endpoints used by the service provider in the resource type endpoint. The primary reason for having dedicated endpoints is so that the service provider can define what HTTP methods can be used in each endpoint. The HTTP methods and their usage is shown in table 4.2

HTTP Method	SCIM Usage
GET	Retrieve a resource
POST	Typically used to create a resource, but can be used for search request and to bulk-modify resources
PUT	Modification of resources where applicable. Cannot be used for creating a resource
PATCH	Modification of certain attributes in a resource.
DELETE	Used for deleting resources

Table 4.1: HTTP methods used in SCIM [3]

SCIM uses these HTTP methods as the only way to exchange data between the service provider and the client. The GET method is used for retrieving data from a system. A regular case is when a client requests updated data from a service provider. The POST method is used for creating new resources and can only create one resource at a time. The POST method can be used to modify the database by hand and also to create new resources in a client application e.g. when a resource is granted access to a new application. The DELETE method is in all it's simplicity used to delete resources from the SCIM service. The deletion of data is done by the service provider so that it can ensure all the provisioned applications receive the call.

The PUT and PATCH operations are both made for the modification of an existing resource. The difference between these two methods have is that the PUT method is made to overwrite the entire resource and the PATCH to only change the selected information.

The PUT method is used to modify single resources in SCIM. Additionally, the PUT method has to be implemented in SCIM and the PATCH method does not. The PATCH method is more precise in what can be added to the databases. The PATCH method can be used to add single users into groups without having to rewrite the whole resource in the database. This kind of method does require more code to function, but saves time in database fetches and saves. These five HTTP methods are what the endpoints are called within SCIM.

The endpoints in SCIM are User, Group, Self, Service provider config, Resource type, Schema, Bulk, and Search. They all have different rules on what HTTP methods they accept. The user endpoint format is /Users and it accepts all the HTTP methods that are defined in the SCIM protocol. This is the same for the group endpoint, where the format is /Groups. The self endpoint is made so that an authenticated user can check their data by using the /me endpoint, which also accepts all the HTTP methods. These endpoints are the most used when handling identity in SCIM.

The rest of the endpoints are more supporting endpoints such as the service provider config where the client can only use a GET method to get the configuration of the service provider. The endpoint format is /ServiceProviderConfig. The resource type, with the format of /ResourceTypes, also accepts only a GET method and it responds with the supported resource types within the service provider. The schema endpoint is /Schemas and it also accepts a GET method to retrieve the supported schemas the service provider has.

The Bulk and Search endpoints have a format of /Bulk and /.search. These endpoints only accept a POST method. The bulk endpoint is there for adding a large number of users or groups into the system. The search endpoint is to search the database of a known endpoint e.g. /users/.search will search the database and return the wanted value. The endpoints are created so that all client know what URI they need to call for certain information in SCIM. Next, the aim is to create a standardized way of creating, reading,

updating, and deleting the resources through the known endpoints using the HTTP methods.

4.3 Creating Resources

The creation of resources is necessary to get the identity management system ready for use since there is no point in keeping an empty system. Although usually in the real world the initial provisioning will happen with a Bulk request, this will be looked at later. The main concept of creating resources in SCIM is to send an HTTP POST message to the service provider with the attributes the resource should have. In most cases, the rules that are in SCIM should be enforced by the service provider. Although the client system should also abide by the rules because the service provider can create users in their systems. The rules documented in the SCIM protocol are as follows: [3]

- The protocol states that "attributes whose mutability is "readOnly" shall be ignored".
- Read-write attributes that are not included in the received message body can be added by the service provider. This means the service provider may add a default value to the attributes.
- The service provider can check if the client has access to all the attributes that are not included in the message body, to determine if the attributes should be added with a default value.
- Clients can override the service provider defaults with adding a null value in the message body when creating a resource. This means using a "null" value in single attributes and an empty bracket "[]" in case of multi-valued attributes.

The rules stated in the protocol helps the service provider and client have the same ways of accepting data and helps avoid conflicts. The protocol also stated some return

codes for the HTTP messages. When the creation of a resource is successful in the target application the target should respond with an HTTP status code of 201 (Created) and add the resource to the response body. This ensures the synchronisation of the data between the two services. Additionally the URI of the resource in the target can be added into the location header in the HTTP response. If the service provider encounters an error in the received data e.g. a duplicate username the service provider will return an HTTP code of 409 (Conflict) and error code "uniqueness" which is a SCIM type error code, these codes will be examined in another section. [3]

Creating resources is very simple in SCIM, but it is necessary to follow the rules because if something basic is messed up the whole system will not work properly. I would say that SCIM is working as a baseline for applications to create at least the most simplistic identity management system. In already existing applications SCIM helps to simplify the systems processing of data with these simple rules.

4.4 Retreiving Resources

Retrieving resources is done with the HTTP GET method and this is done so the data can be used in different situations e.g. showing a list of all users in the database. The service provider is responsible for checking what attributes will be returned, these are the attributes that have a returned characteristic of always or default. These characteristics are defined in each resource's schema for reference see section 3.1. [3]

The protocol addresses three different ways of retrieving resources the first being retrieving a known resource. This is the basic way of getting a resource where the ID is known and there is only one resource the client wants to access. This happens with an HTTP GET call. The call is sent to a resource endpoint and the id of the resource is appended into the URI e.g. /Groups/{id} to retrieve a known group. When this kind of retrieval is done the service provider responds with an HTTP 200 (OK) status and the re-

source that was asked for. If the resource is not found in the database the service provider may return an HTTP 404 (Not Found) status to the client. [3]

The two other methods of retrieving resources are query and query with HTTP POST these two methods have multiple different ways of operation and for the sake of page management this section will not cover them fully, for further reference please see RFC 7644 pages 15-28 for full information [3]. Querying with an HTTP GET method is that the client can send queries to the resource endpoints with different functionalities such as a filter, sort, and paginate. These queries are returned in a ListResponse format. This format includes total results, resources, start index, and items per page. The total result means all the hits a query gets. Resources is a multi-valued list of the found resources. The start index is the index of the first result in the current set of resource, this is only used when pagination is done. Items per page show the number of resources on one page. [3]

Filtering is one way to use the query and it uses the ABNF [21] on how the operators used in filtering are read. The idea is to use the keyword filter in the URI when filtering is applied to the endpoint e.g. baseuri/v2/Users?filter=userName Eq "Bob". This query will search the Users endpoint for all resources with a username of Bob. All the query keywords can be found in the SCIM protocol RFC 7644 [3]. Filtering enables searching the database for a specific resource simply and efficiently . [3]

Pagination is used to separate a large number of resources into readable chunks of information. This is useful when it is important to list a lot of resources. Sorting is optional for the service provider, but if chosen to be implemented the sorting uses two keywords of sortBy and sortOrder, which determine how the information should be listed. The sort by keyword can determine what attributes the sorting should be done by. The sort order defines if the order is ascending or descending on the list. [3]

The querying with the HTTP POST method is done with a /.search keyword at the end of the URI, this will inform the service provider the incoming message body will have the following information; attributes, excludedAttributes, filter, sortBy, sortOrder, startIndex,

and Count. These values determine what attributes are queried and what filter is used with the attributes. This is an alternative way to query data from the database. [3]

These are the three ways of retrieving resources in SCIM protocol the main way would be to get resources that are known by the client. The querying is very complex to create in real life and must be done with care as to not have something wrong returned. The query is very useful when implemented and helps with getting resources. [3]

4.5 Modifying Resources

Modifying resources can be done two ways in SCIM with the PUT and PATCH methods. The main difference in PUT and PATCH is that PUT replaces everything and PATCH only replaces given values. First, we will look at the PUT operation and after the PATCH. Replacing attributes with PUT means overwriting everything on the service provider side unless the attributes characteristic is read-only or immutable. In the latter case, the client can send immutable attributes, but they must have the same value as the server-side attributes if this is not the case the service provider will send an HTTP status code of 400 (Bad Request) and the error code of mutability. [3]

The biggest and most important thing that the service provider must check is that it is not possible to create new resources with the PUT operator. The client may send a full representation of the resource and not care about the mutability. When the PUT operation has been successfully done the service provider will send a 200 (OK) code to the client with the updated resource in the body. [3]

Modifying the resource with a PATCH operation is an optional feature in the SCIM protocol. This is usually still implemented because it helps to update single users in a group rather than always having to update the whole group. The body of the SCIM PATCH operation uses the HTTP patch format shown in listing 4.1. The idea is to have the operations as an array that contains the operation that should be done, the path where

the operations should be done, and the value that is going to be changed. The listing 4.1 shows the addition of a new member into a group. The patch operation can have multiple operations and all of the operations will be evaluated one by one until there are no more operations or there is an error code. [3]

The Patch operation has more rules than the more simple PUT operation and now we will discuss them in detail. Before looking at all the possible operations in the PATCH call we will look at how the service provider will determine the path. The path attribute is a simple string that is optional when sending add and replace operations and required with the delete operation. The idea is that the path determined the exact attribute that we want to add or modify e.g. if the operation was replaced and path name.familyName then the value of the operation would be meant to replace the family name of that resource. The path operations ensure that the operation will happen to the right attribute. [3]

Listing 4.1: PATCH format

```
{ "schemas":
    ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
    "Operations":[
    {
        "op":"add",
        "path":"members",
        "value":[
            {
              "display": "Babs Jensen",
              "$ref":
    "https://example.com/v2/Users/2819c223...413861904646",
              "value": "2819c223...413861904646"
            }
        ]
```

```
15      },
16      ... + additional operations if needed ...
17      ]
18  }
```

The operation attribute in the PATCH request determines what should be done. The three operations possible are add, replace, and delete. The add operation is used to add a new attribute value to an existing one. The operation must contain a value which can be a single value or a multi-valued JSON object. This makes the add method usable in all cases, which are specified in the protocol and depend on the path. These rules are taken from the SCIM protocol [3].

- If omitted, the target location is assumed to be the resource itself. The "value" parameter contains a set of attributes to be added to the resource.
- If the target location does not exist, the attribute and value are added.
- If the target location specifies a complex attribute, a set of sub-attributes SHALL be specified in the "value" parameter.
- If the target location specifies a multi-valued attribute, a new value is added to the attribute.
- If the target location specifies a single-valued attribute, the existing value is replaced.
- If the target location specifies an attribute that does not exist (has no value), the attribute is added with the new value.
- If the target location exists, the value is replaced.
- If the target location already contains the value specified, no changes SHOULD be made to the resource, and a success response SHOULD be returned. Unless other

operations change the resource, this operation SHALL NOT change the modify timestamp of the resource.

These rules must be followed when implementing a SCIM application that uses the PATCH operation. The implementation part of this will be discussed in the next chapter. The next operation that the PATCH method uses is replace which is meant to replace the value at the specific path mentioned in the message body. Similar to the add operation the replace has rules that it must conform to. These rules are taken from the SCIM protocol [3].

- If the "path" parameter is omitted, the target is assumed to be the resource itself. In this case, the "value" attribute SHALL contain a list of one or more attributes that are to be replaced.
- If the target location is a single-value attribute, the value of the attribute is replaced.
- If the target location is a multi-valued attribute and no filter is specified, the attribute and all values are replaced.
- If the target location path specifies an attribute that does not exist, the service provider SHALL treat the operation as an "add".
- If the target location specifies a complex attribute, a set of sub-attributes SHALL be specified in the "value" parameter, which replaces any existing values or adds where an attribute did not previously exist. Sub-attributes that are not specified in the "value" parameter are left unchanged.
- If the target location is a multi-valued attribute and a value selection ("valuePath") filter is specified that matches one or more values of the multi-valued attribute, then all matching record values SHALL be replaced.
- If the target location is a complex multi-valued attribute with a value selection

filter ("valuePath") and a specific sub-attribute (e.g., "addresses[type eq "work"].streetAddress"), the matching sub-attribute of all matching records is replaced.

- If the target location is a multi-valued attribute for which a value selection filter ("valuePath") has been supplied and no record match was made, the service provider SHALL indicate failure by returning HTTP status code 400 and a "scimType" error code of "noTarget".

These rules give the basis of how the replace operation should work. There are multiple different ways the path and filter affect the operation and the rules above cover all of these situations. These will be looked at more in the implementation part. The last operation that the PATCH method uses is remove. This is meant to delete attribute values. These rules are taken from the SCIM protocol [3].

- If "path" is unspecified, the operation fails with HTTP status code 400 and a "scimType" error code of "noTarget".

- If the target location is a single-value attribute, the attribute and its associated value is removed, and the attribute SHALL be considered unassigned.

- If the target location is a multi-valued attribute and no filter is specified, the attribute and all values are removed, and the attribute SHALL be considered unassigned.

- If the target location is a multi-valued attribute and a complex filter is specified comparing a "value", the values matched by the filter are removed. If no other values remain after removal of the selected values, the multi-valued attribute SHALL be considered unassigned.

- If the target location is a complex multi-valued attribute and a complex filter is specified based on the attribute's sub-attributes, the matching records are removed. Sub-attributes whose values have been removed SHALL be considered unassigned. If

the complex multi-valued attribute has no remaining records, the attribute SHALL be considered unassigned.

The remove operation is not meant to delete resources, but rather delete specific attributes values that is why the path is mandatory in the remove operation. Other than that the values are removed with the specified rules. The PATCH operation is more complex than the other HTTP methods used in the SCIM protocol. [3]

The PATCH method gives the freedom of efficiently changing single attributes values where the PUT method would need to overwrite a big chunk of data. Implementing the PATCH is a good idea in most cases, just because it will add value to the modification and simplicity in working with groups.

4.6 Deleting Resources

Deleting resources is used to do exactly what the name states to delete the resource. This is a simple HTTP method called DELETE sent by the client with the information of a resource id. The service provider will delete the resource or it can keep the data but must return a 404 (Not Found) every time the client tries to retrieve the data. What should be remembered is when deleting resources that if they are connected to other resources that also the other resources are updated e.g. a user is deleted the affected groups should also be updated. [3]

When the service provider deletes a resource it needs to send the DELETE method to all the client applications that have a copy of the resource. This ensures that the lifecycle of resources ends. If this is not done the user could have access to existing applications with old credentials creating a big security risk. [3]

4.7 Additional operations and endpoints

Status	Usability
307 (Temporary Redirect)	GET, POST, PUT, PATCH, DELETE
308 (Permanent Redirect)	GET, POST, PUT, PATCH, DELETE
400 (Bad Request)	GET, POST, PUT, PATCH, DELETE
401 (Unauthorized)	GET, POST, PUT, PATCH, DELETE
403 (Forbidden)	GET, POST, PUT, PATCH, DELETE
404 (Not Found)	GET, POST, PUT, PATCH, DELETE
409 (Conflict)	POST, PUT, PATCH, DELETE
412 (Preconditioning Failed)	PUT, PATCH, DELETE
413 (Payload Too Large)	POST
500 (Internal Server Error)	GET, POST, PUT, PATCH, DELETE
501 (Not Implemented)	GET, POST, PUT, PATCH, DELETE

Figure 4.1: HTTP status codes used in SCIM

The SCIM protocol is quite extensive and the scope of the thesis is not to cover every aspect of the protocol so this chapter will cover other parts of the protocol that have not been discussed detail. This will include the /ME endpoint, bulk operation, HTTP status and error codes, and SCIM versioning. To start of the "Me" endpoint is an optional endpoint that is used when the client is authenticated. The endpoint will get the authenticated client information and all the resources that authenticated client is connected to. This is could be considered as a personal information page in a web browser. [3]

The bulk method is another optional feature the service provider can have and it is used to handle a large number of resource additions at once to the service providers database. Bulk operations are handled with a POST call to the /bulk endpoint withing the SCIM application and it handles operations is the same way as PATCH does. The difference is that the bulk operations define the method e.g. the HTTP method in the operations. For full information about the bulk operation please refer to RFC 7644 pages 49-63 [3].

The HTTP status codes are a crucial part of SCIM messaging. The SCIM protocol uses the status codes defined in RFC 7231 chapter 6 [19]. The SCIM protocol also states that in addition to sending the status code the error messages must be sent as JSON format in the body of the message, these errors are illustrated in figure 4.7. The protocol defines that the error code of 400 should include a scimType error message in the body, these messages and descriptions are shown in figure 4.2. Using these messages the error handling is much easier than with only using error codes.

The last thing discussed in this section will be the SCIM protocol versioning. This is relevant because SCIM aims to develop in the future and the version controlling will allow clients to see what version is in use. The version will be added to the URI of the SCIM application. In SCIM 2.0 the version identifier is added as follows baseurl/v2/endpoint here the "v" stands for version and the number after is the iteration in use.

scimType	Description	Applicability
invalidFilter	The filter syntax is incorrect, or the attribute and filter comparison is not supported	GET, POST (search), PATCH (path filter)
tooMany	The filter retrieves more resources than what the service provider is able to hand out	GET, POST (search)
uniqueness	The attribute value is already in use	POST (create), PUT, PATCH
mutability	The modification of the value is not possible due to the attribute characteristic	PUT, PATCH
invalidSyntax	The message body was not in SCIM standard	POST (search, create), BULK, PUT
invalidPath	The path attribute is not in correct form	PATCH
noTarget	The target of the path does not exist e.g. when a filter does not give matches	PATCH
invalidValue	The value is missing or the value was not compatible with the attribute type	GET, POST (create, query), PUT, PATCH
invalidVers	The version of the SCIM protocol is incorrect	GET, POST (all), PUT, PATCH, DELETE
sensitive	The request cannot be done, because sensitive information must not be passed in a request URI	GET

Figure 4.2: The SCIM status 400 error messages.

4.8 Service Provider Configuration Endpoints

This section will cover the endpoints mentioned in section 4.2. All of these endpoints include configuration information about the service provider, the endpoints are /service-PorviderConfig, /Schemas, and /ResourceTypes. The service provider config endpoint is meant to return a JSON message that describes the SCIM specifications that are in use by the service provider. The Schemas endpoint can be called by an HTTP GET and it will return all the schemas that are in use by the service provider. The Schemas will be returned in the list response form.

Resources types endpoint is provided so that a client can retrieve information about all the resource types that are supported by the service provider. The resource types define the endpoint, core schema URI, and any supported schema extensions. These three configuration endpoints give the clients an idea of what calls and resources they can use. Usually, the service provider does, however, provide more documentation to the clients if they need it, but in theory, the endpoints should provide all the necessary information.

4.9 Summary of the SCIM Protocol

The SCIM protocol complements the core schema by taking the resources and creating specifications on how they are used. The protocol is not meant to be a guide on how a SCIM application should be implemented and therefore it does not address security implementations in detail. The protocol does, of course, state the need for security such as authentication and authorization, security and data storing, HTTP, and TLS support. The basic idea of what the protocol gives about security is that implementations should the best method of security at the given time. At the time of writing this thesis, the currently most used authentication protocol used in REST API's is the OAuth2.0 [22], it is also the suggested authentication method in the SCIM protocol. The fact that the protocol does not limit the security features allows the use of this protocol even when the

methodologies have changed making the protocol more durable to aging.

SCIM uses HTTP methods to transfer data across the multiple domains it has. This is made possible by having one service provider that handles all the client's HTTP calls and is responsible for having the "truth" in its database. The clients can vary from single users that send HTTP calls or another system that uses SCIM messaging to provision itself. The SCIM protocol also takes a look at multi-tenancy, which means that a single service provider can have multiple clients where the service provider makes sure the data is not accessible by other clients even though they are stored in the same database. The addition of multi-tenancy makes the SCIM protocol more versatile and means setting up new clients doesn't take too much time. The protocol document compared to the core schema takes more real-life situations into account and it gives implementers a good starting point.

The next chapter will be an implementation of real SCIM applications that consist of a service provider and a client. The idea is to create a testing tool that utilizes purely SCIM messaging and can be connected to already operational SCIM applications. The chapter will also discuss the implementation process and the drawbacks that the SCIM protocol and schema might have.

Chapter 5

Project SCIM

Project SCIM is a testing tool written in C# [23] for the back-end and Angular 8 [24] for the front-end, the database used during development was SQL Server Express [25]. The actual platform that was used in creating the back-end is the Microsoft .NET Core [26]. Testing of this application was done using Azure Active Directory [4] to do provisioning to the client application, this will be shown in the latter part of this chapter. This chapter will feature a new SCIM testing application and discussion about the benefits and drawbacks of SCIM. The discussion that this chapter will include is the thoughts of the author and does not reflect the practices of Fujitsu Finland Oy.

The reason why this project was done from scratch and not using already existing open-source implementations is that Fujitsu required a testing tool that could be used both as a client and service provider. The existing open-source implementations are only either a service provider or the client, also most don't have proper documentation. There was also the fact that the code would be best written in .NET and the frontend with Angular so that we would have the same code basis for our systems. These factors made it clear that the best solution was to build two different SCIM applications from scratch using the .NET Core and Angular. The goal of the project was to have two SCIM applications with user-interfaces (UI) called the source and target, where the source is the service provider and the target is the client, these names will be used in this chapter.

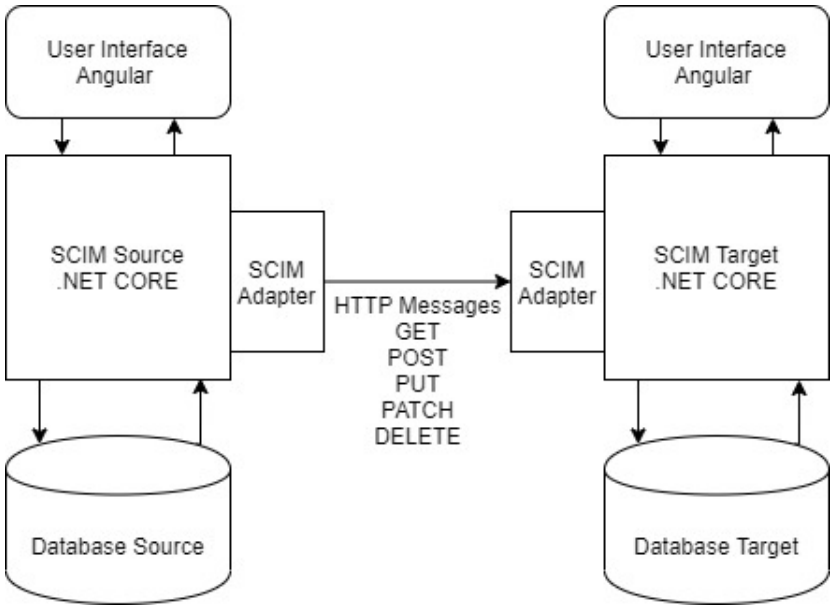
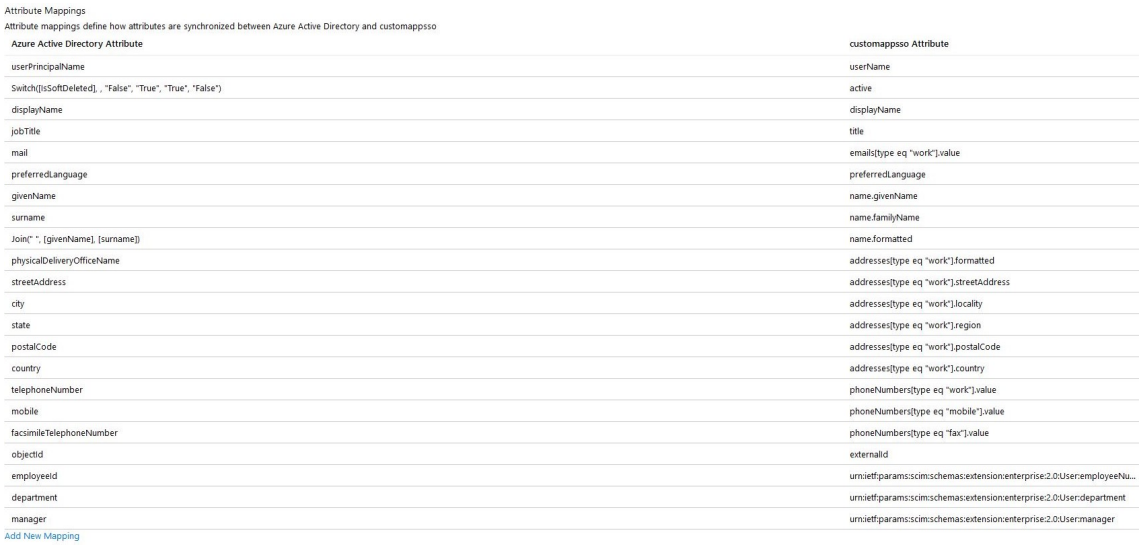


Figure 5.1: Model of the SCIM project

5.1 Scoping of the project

The project was scoped with the help of the Lead System Architect from the IAM team. As mentioned the project should include a working source and target that both have UI. The reason why both the source and target should be made is that the testing tool can be used for checking the whole loop that the data will travel in the real system and see if there are changes that should not happen. To start the project a model was made so that it is more simple to understand what is needed, the model is showcased in figure 5.1. The two different SCIM applications are divided and they work separately with their respective UI, but the real idea is to have the source provision the target with users and members with HTTP calls. This differs from the SCIM protocol in a way that the service provider usually receives the HTTP calls and works accordingly, but this way the source can provision multiple real-world applications with user data to create accounts.

Figure 5.1 also has something called the SCIM adapter, what this means is that SCIM should be considered as an adapter that can be built on top of already existing applications. In this project, the ability to use SCIM as an adapter is not showcased because the


Attribute Mappings
Attribute mappings define how attributes are synchronized between Azure Active Directory and customappsso

Azure Active Directory Attribute	customappsso Attribute
userPrincipalName	userName
Switch([IsSoftDeleted], , "False", "True", "True", "False")	active
displayName	displayName
jobTitle	title
mail	emails[type eq "work"].value
preferredLanguage	preferredLanguage
givenName	name.givenName
surname	name.familyName
Join(" ", [givenName], [surname])	name.formatted
physicalDeliveryOfficeName	addresses[type eq "work"].formatted
streetAddress	addresses[type eq "work"].streetAddress
city	addresses[type eq "work"].locality
state	addresses[type eq "work"].region
postalCode	addresses[type eq "work"].postalCode
country	addresses[type eq "work"].country
telephoneNumber	phoneNumbers[type eq "work"].value
mobile	phoneNumbers[type eq "mobile"].value
facsimileTelephoneNumber	phoneNumbers[type eq "fax"].value
objectId	externalId
employeeId	urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:employeeNu...
department	urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:department
manager	urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:manager

Add New Mapping

Figure 5.2: Azure AD user mappings


Attribute Mappings
Attribute mappings define how attributes are synchronized between Azure Active Directory and customappsso

Azure Active Directory Attribute	customappsso Attribute
displayName	displayName
objectId	externalId
members	members

Add New Mapping

Figure 5.3: Azure Ad Group mappings

backend is built to resemble SCIM resources, but with already built identity management applications SCIM could be used as an adapter. It would require mapping the SCIM attributes to already existing attributes, but it saves time and money, for example, Azure AD uses this type of adapter as shown in figures 5.2 and 5.3. This type of SCIM adapter is a good idea if all of the applications use the adapter, but when that is not the case it is not worth the effort. In a perfect environment, all the applications that are managed would use only one standardized protocol for communication, which SCIM aims for.

Both the source and target have their databases which are modeled based on the SCIM resources mentioned in the Core Schema [2]. Figure 5.4 illustrates a simplified version of how the databases are modeled. This was a rough sketch made during the scoping of the project, but it gives a good example of what the databases should contain. Figure 5.4 also explains how the source and client will determine their ID values to each resource,

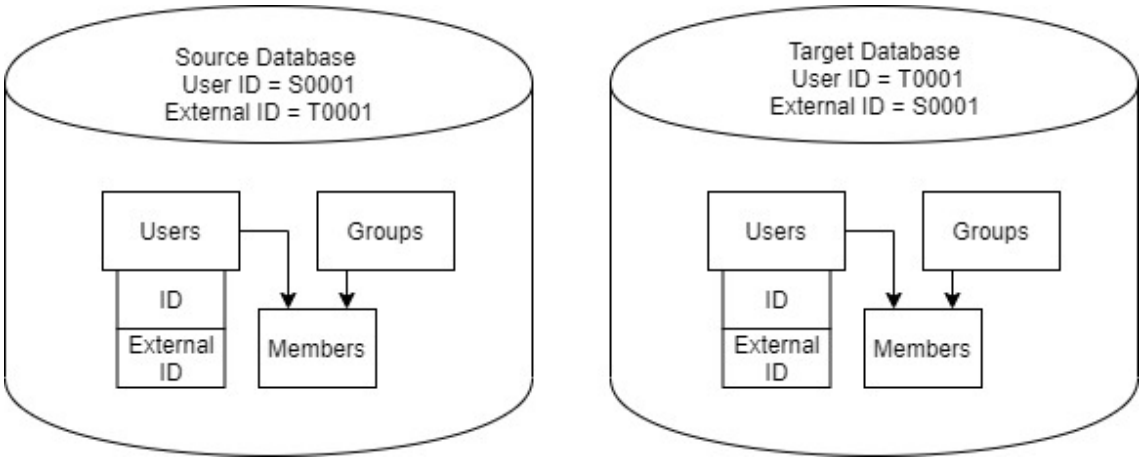


Figure 5.4: Source & Target Model

where the source ID will be the external ID of the target and vice versa. Figure 5.4 also shows the data binding that the resources will have, meaning both Users and Groups will be connected to the members the reason why this is done as such will be discussed in a further section about the back-end implementation.

The planning stage was quite extensive in this project since I had to study about SCIM, .NET Core, Angular and Microsoft SQL. The good thing about this was that I was able to make a clear plan on what to start doing first and then move forward. The actual process that I planned out was to first create the back-end of the source and then build a UI for it. After that was done the idea was to copy the basics to the target application and then change the necessary code. This was possible because the databases would be identical and the API would use the same calls. Other than that the code was customizable. After the actual coding was done the solution is tested with creating a docker image of the target application and using Azure Active Directory to provision test subjects into the target. The test would confirm that the target was working with actual SCIM messaging with a real application.

5.2 The Back-End

The back-end was first built for the source but then copied over to the target. This section will cover the coding included in the source and then the changes that were made to the target. Before we can look at the actual project we must understand what the .NET Core platform is about. The .Net Core runs on the three most used operating systems meaning Windows, Linux, and macOS. This was a key reason why it was chosen to be the basis of the back-end. Docker images are run in Linux kernel and this means that the code must be compilable in Linux. The .NET Core uses C# language for the code that it compiles. Other than that this thesis will not cover, because there are good documentations for the .Net Core online. [26]

The project was started with a .NET Web Application that creates a basis that I could build upon. The template contains the basic files required to run such as the Program, Startup, and an example ValuesController. The controller is where the HTTP calls are caught and then handled appropriately. The values controller does not do anything thus it was deleted. The changes made to the startup files are minor but necessary. The startup file is where the database connection is configured also the cors policy was updated to obtain testing properly. Creating models in .NET Core is the focal point as they will create the resources into the database. The coding practice used in this application is code first method where the models are created in the code and then the models are used to create the database. This requires having a context that works as the bridge between the application code and the database.

5.2.1 Database Context and Models

The Context represents a session with the database and it supports the query and saving of entities when used in the code. This means any changes that need to be done to the database requires a connection using the database context. These are stated in the con-

troller and will be discussed in the next sub-section. The context in the source defines all the entities that need to be addressed in the database. These can be seen in listing 5.1 the entities are created using the DbSet<...> where the ... is the name of the model created in another file.

The actual creation uses the onModelCreating method where the entities are mapped into tables. In the SCIM protocol, the only resources defined are user, group, enterprise user, and resource. In this project, more entities are created and handled in the database due to the nature of the .NET Core framework. The reason is that the models are objects and to get the desired JSON format the objects need to be done in the correct format. This means that if an attribute is multi-valued the then the attribute must be an entity itself and have a relationship with the main entity. In .Net Core this is done with keys that are defined in the OnModelCreating method, this can be seen in listing 5.1 on line 28 where the address entity is tied to the user with the keyword HasOne. Additionally, the relationship is built with defining that the user can have many addresses and the deletion behavior is cascade, meaning that when the user is deleted so are the addresses tied to that user entity.

Listing 5.1: Database Context

```
1 { public class ScimContext : DbContext
2     {
3         public ScimContext(DbContextOptions<ScimContext>
              options)
4             : base(options) { }
5
6         public DbSet<User> Users { get; set; }
7         public DbSet<Group> Groups { get; set; }
8         public DbSet<Resource> Resources { get; set; }
9         public DbSet<UserGroup> UserGroups { get; set; }
```

```
10          public DbSet<Address> Addresses { get; set; }
11          public DbSet<Email> Emails { get; set; }
12          public DbSet<Entitlement> Entitlements { get; set; }
13          public DbSet<Ims> Ims { get; set; }
14          public DbSet<PhoneNumber> PhoneNumbers {get; set;}
15          public DbSet<Photo> Photos { get; set; }
16          public DbSet<Role> Roles { get; set; }
17          public DbSet<X509Certificate> X509Certificates { get;
               set; }
18          public DbSet<Meta> Metas { get; set; }
19
20          protected override void OnModelCreating(ModelBuilder
               modelBuilder)
21          {
22              modelBuilder.Entity<User>().ToTable("User");
23              modelBuilder.Entity<Group>().ToTable("Group");
24              modelBuilder.Entity<Resource>().ToTable("Resource")
                   ;
25              ...
26
27              modelBuilder.Entity<Address>(e =>
28              {
29                  e.HasOne(x => x.User)
30                      .WithMany(x => x.Addresses)
31                      .OnDelete(DeleteBehavior.Cascade);
32              });
33          }
```

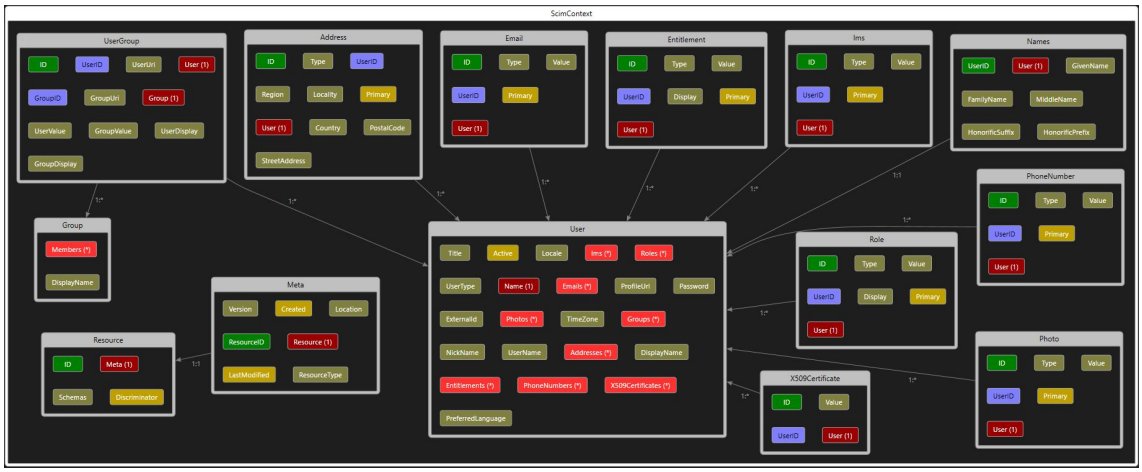


Figure 5.5: Database context of the source

Using the relationship model the database context was created into a working solution, figure 5.5 is the visualization of the relationships the database entities have. Here we can see that the resource entity is entirely on its own and without any connection other than the meta, this is because the user and group entities are inherited from the resource so that they have the needed attributes which are id, meta, and schemas. The Group entity does not have that many attributes inside it because the .NET Core does not support many to many relationships that is why the usergroup entity is created. The usergroup entity is a join table where both the user entity and group entity has access to, that is why there are two sets of Uri, Value, and Display. The user set contains information of the user that is in the specific group and the group set contains the information of the group that the user is in, this way it is possible to link the group and user together with having multiple users in groups and multiple groups in users creating a many to many relationships.

The user entity has multiple one-to-many relationships that can be seen in figure 5.5, all of these entities are multi-valued complex attributes that the SCIM protocol [3] defines. This is so that when the user object is made into a JSON object the format is correct. The names entity is the only one-to-one relationship in the context because it is the only complex single-value attribute in the user resource defined in the core schema [2]. The project that was made does not include the enterprise user since it was not needed during

the initial development and will be added in a later version. This concludes the context part of this section and next, we will look a bit more closely at the user and group entities.

The models in the source are objects that define what attributes are in them, this is normal for strongly typed languages such as C#, this helps in building the database but also to handle the data that comes in from the HTTP messages. The resource model in the source contains the common attributes defined in the core schema [2] these common attributes can be seen in the figure 5.5 and in section 3.2. The meta attribute is an entity of its own because it is a complex-type. Because the meta is connected to the resource entity when the resource entity is inherited the meta is also inherited.

The group model is very simple in terms of its structure it has the display name and list of members. The member list is the join table usergroup which has the values already mentioned. The reason why the group model does not have anything else is that it inherits the common attributes from the resource object. The user model has more objects in it because the core schema [2] defines more attributes. The object can be seen in figure 5.5 and they are all located in the same model in the code.

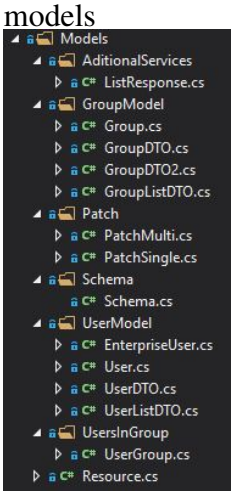


Figure 5.6: The models

There are also additional models in the source program. These models are variations of the actual resources defined in the core schema [2]. The moddels are called data transfer objects. The reason why I made data transfer objects was to have a way of transferring only the necessary data to the UI, also the SCIM documentation states that the password should never be returned and the code returns everything in the object if nothing is done. That is why there are multiple different models as seen in figure 5.2.1. All of these models are necessary for making the whole project work and in the future, there will be more models. The enterpriser user and schema models are not used in this stage of the project but will be implemented if needed. This was something decided by me and my mentor to ensure that

the project will be done in a reasonable time and have it working for testing. The target application also uses the same models because the models are done with following the SCIM core schema [2].

5.2.2 Controllers

Controllers are the application program interface (API) this means that all the HTTP methods and actual processing of the data are done in these controllers. The source and target both have two different controllers called user controller and group controller the reason why these two controllers are separate is that the endpoints are defined with the controllers. It was more simple to have the groups and users endpoints in the separate controllers. This is also where there should be more endpoints supported as stated in the SCIM protocol [3], but it was obvious that it would take too much time to perfect everything so they will be added in the next version.

The first thing that was built was the user controller, it uses the SCIM context mentioned in the previous section for transferring data between the database and the HTTP client factory to receive HTTP calls. The actual methods that require sending over data use HTTP requests to send the data. The user controller for the source and target both have all the HTTP methods that are required by the protocol; GET, POST, PUT, PATCH, DELETE. These HTTP methods all have their classes written in the code that uses some value received from the HTTP message body. The great thing about the .NET Core is that it can map JSON messages straight into objects if the object has a model defined, this makes it super easy to read the messages and check the data that was received. This functionality is really good for something like SCIM where you know the type of message you should receive. The bad part is that if the format of the JSON message has even one mistake the message won't be read properly.

The source user controller has some additional methods that are used to communicate with the UI, these have nothing to do with SCIM messaging, but it was necessary to get

the UI working. The methods that are used in the SCIM messaging all have some things in common they all receive a JSON format user that has values, except for the GET method which receives either nothing or then an ID in the URL. This means that it is possible to get a list of all the users in the database or then retrieve a single user from the database using the ID.

The code has two different GET methods that are separated by the endpoints the regular GET is done called on the "baseurl/api/v2/users" endpoint and the retrieval of a known resource is done to the "baseurl/api/v2/users/ID" endpoint where the ID is the id of the known resource. The GET method for retrieving all the users is not something mentioned in the protocol, but it was used to test that all the users are added when testing. It is not used in the communication between the source and target. The biggest problem that was faced during the coding was the functionality of the query because .NET Core does not support comparing a string with the object name and thus it was necessary to have a workaround that is most likely not the best. That is why the query is not completely working in this project and it was decided it is not necessary for this version of the project.

The POST method simply receives a JSON message that includes user data. The method first checks that the username that must not be empty and that it is unique in the database. If these two things are fulfilled then the POST method reads the JSON object and creates a new user entity and maps the values to the correct place. After the new user entity is created it is sent over to the target application so that it can create the same resource and return an OK message. When the target has responded with the OK message the entity is saved into the database and the information is sent back to the UI so it can display a created message. This is done because the source application uses HTTP messages between the UI and back-end, this workflow can be seen in figure 5.7. The only difference between the target and source POST methods is that the source will send out the HTTP POST to the target which is true in all the other HTTP methods also.

The PUT method used in the source is for adding a whole user and overwriting the

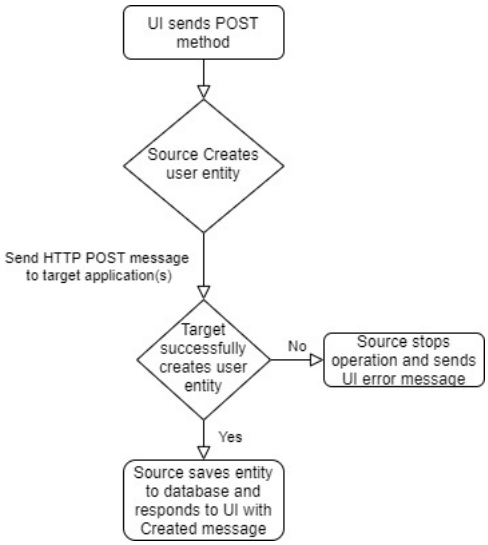


Figure 5.7: HTTP POST Workflow

attributes that are allowed. This is done by receiving a JSON style user message and checking that the message is not empty. The call is made to a certain user endpoint so the id is used to fetch the user from the database. Then the values are overwritten to that user entity and saved. As in the POST method also the PUT method sends the request forward to the target. The PATCH method uses the same workflow as the other methods. The implementation of the PATCH method was the hardest in the whole project and required over 500 lines of code to get the replace operation working. This might be because of my lack of expertise in coding, but also because it is not possible to compare a string value with the name of an attribute e.g. if the value is familyName you cannot compare it to name.familyName. Therefore it is necessary to check all the possible solutions there are.

The DELETE method was the most simple thing to code and it works in just calling the HTTP DELETE method to the endpoint of a known user and then the user will be deleted from the database. The user controller contains two more additional GET methods that are count and list and they are used to convey information to the UI. The target application has the same methods in it, the difference is that it does not send any HTTP requests to any other application e.g in this case to the source.

The group controller has the same HTTP methods that the user controller and they

work in the same way. Adding users to groups is implemented differently in the project, because the UI required it. The UI uses an add user endpoint where the adding of users is handled. This was implemented because at the beginning the PATCH operation was not done and the UI needed testing. That is why the UI does not use the proper PACTH way of adding the users into groups. This is not a problem because what matters in SCIM is only that the messages the service providers receive and send to other SCIM applications are in SCIM format. All the rest in the back-end and front-end does not matter. This is something that I found to be a good thing when implementing the UI into the backend system.

5.2.3 Improvements

The fact that the source and target are working does not mean they are fully operational SCIM implementations with all the required functionalities that the SCIM protocol [3] states. There is much to improve on and some things have already been mentioned. To make it more clear this section will be about the improvements required. The user and group controllers are working and they comply with the standards, but the query option is limited in terms of what is queriable, this is because the testing phase only required an empty query response. That is the biggest thing that should be done in the user and group controller. The next thing would be to add the Service provider, Schema, Bulk and search endpoints. This was left out intentionally because the purpose of the testing tool is to test the flow of user and group data.

Other things that should be implemented in the project are more models, for example, the enterprise user model and then additional models that are used in the system that is tested. This was a learning process for so me there is some code refactoring to do to make the code more readable and compile better. The fact that this was the first .NET Core project that I made I am happy with the outcome of it. The next section will be about the front-end side of the project.

5.3 Front-End

This section will cover the Angular UI implementation, which will rely on the user to have a basic understanding of the Angular framework. This will be a high-level overview of the project UI and will not go into detail about all the components. Both the front-end applications for source and target are the same just that they are calling a different base URL. So this section will cover how the application was built. The Angular framework uses components to build single-page web applications that can be used as application UI's. The data that flows between the front-end and back-end happens with HTTP methods, that is why it was a good option to use angular as the front-end.

Figure 5.8: Dashboard view in the UI

The main focus in this section is on how the pages are built and how they work. The pages are how the application view is built in the browser. The pages can be navigated with the top left corner button in the dashboard as seen in figure 5.8. The Dashboard shows the user how many registered users are in the database and the number of unique groups. The idea is that the dashboard component sends an HTTP GET message to the back-end when the page loads and receives the data about the numbers and then shows

the result to the user. The user can then navigate to view all the users or groups in the database with a list view by clicking the big circles, or then by clicking the top bar buttons. The buttons open up a list view where the user can add or view the users or groups. The Dashboard is very simple because this is only a test tool and does not require loads of functionalities.

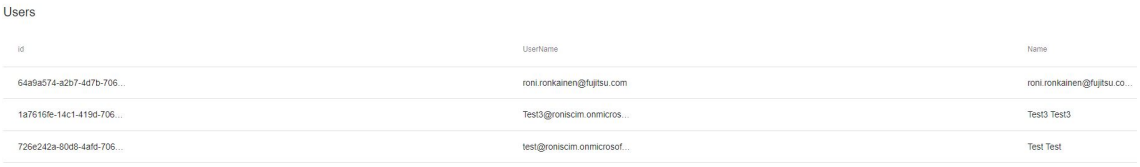


Figure 5.9: Userlist view in the UI

5.3.1 User Pages

The User and Group pages both have more functionalities, but first, we will take a look at the user pages that has three different pages. These pages represent the different views a user sees and they are add user, full user, and user list. Figure 5.9 is the user list page where all of the users in the database are shown, if there are large quantities of users the list will show the first 10. This is done so that the page will be more user friendly. The user list page takes advantage of the additional endpoint not related to SCIM in the backend that returns a user list with only the id, userName, and Name. This is done because it is unnecessary to send all of the user data over, and also when there are a large number of users the view should load faster. This user list page is only there to show all of the users in the system and also create a link to the real user page. The list can be clicked and then the user will be routed to another page where the clicked user information will be shown in SCIM format called the full user page.

The full user page shows the user attributes and enables modification to them. The data is retrieved from the backend with the HTTP GET method using the known id from the user list page. The id is sent to this view when the application user clicks the user

information. The style of the page can be seen in figure 5.10, the attributes are not editable as a default but can be turned to editable with the edit button. This changes all the fields into editable lines expect the group attribute because the users should be added from the group page. The save changes button is used to trigger an HTTP PUT method that sends the edited user to the back-end and then redirects the user to the user list page. The delete button is used for sending an HTTP DELETE message and when the user is deleted redirect the user back to the user list page. The functionality of the UI is very simple, but the necessary SCIM methods are used.

The last page view in the user pages is the add user page where users can be created from the UI. This page uses the HTTP POST method and sends it to the back-end, and waits for a response to verify the user was added. The view of the page is very similar to the full user page because they both use the same component called dynamic forms that generates the view using a schema and data given to it. The view is shown in figure 5.11 where all of the attributes are editable. The user can just type the wanted values to each attribute and then press a save button which will trigger the HTTP POST method to the back-end. There are some attributes such as the id and meta that are not included in the view because they are not supposed to be given values by the users. The multi-valued attributes for example emails have the plus and negative buttons next to them where additional attributes can be added or deleted.

The user pages create the basis for the UI on how to modify the user data in the database. The user pages use a user service file that makes all the HTTP calls possible. The methods are GET, POST, PUT, and DELETE the PATCH method is not used because it was too complex to start checking what fields had been updated and what to send back to the back-end, the PUT method works fine with modifying user data.

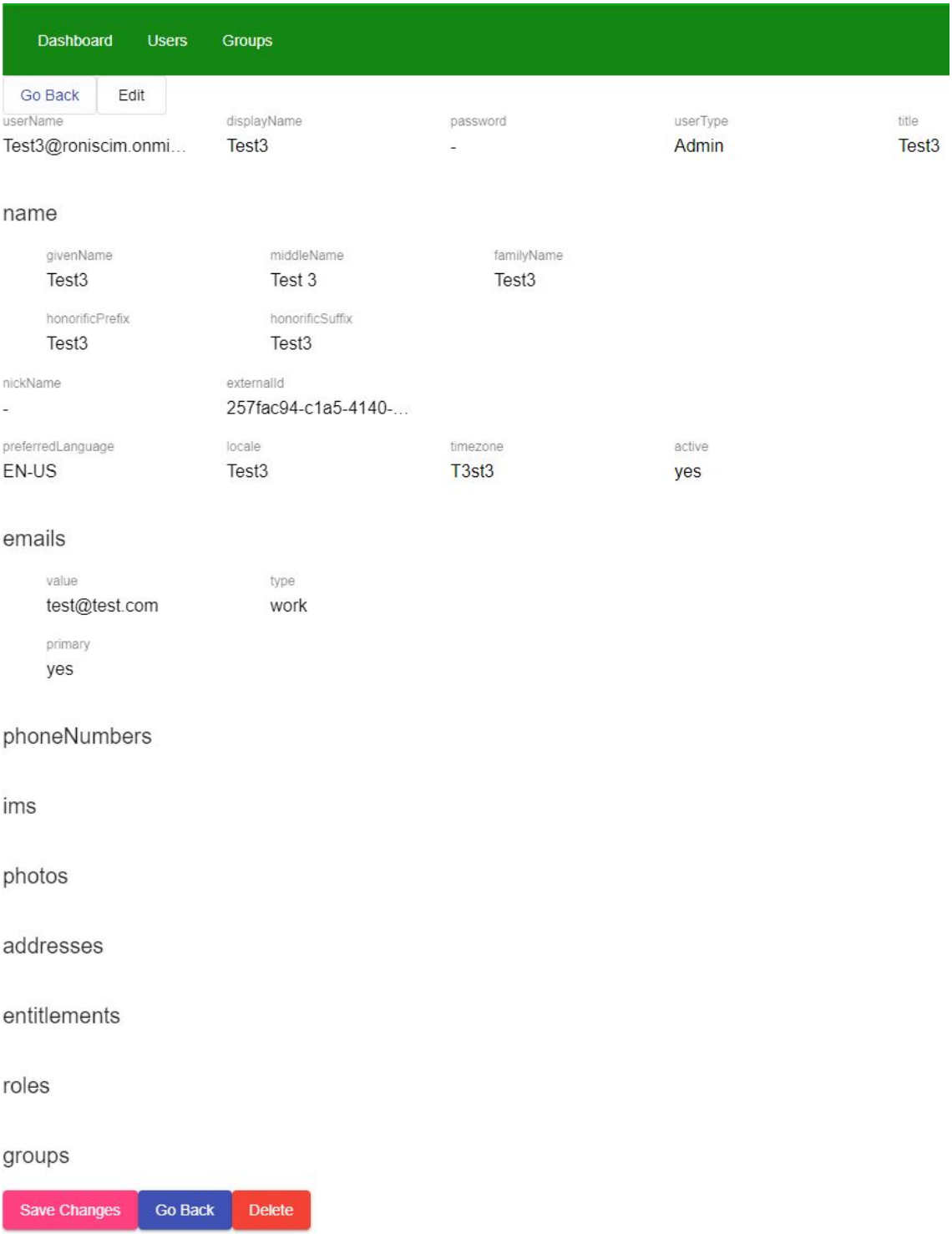


Figure 5.10: Full user page

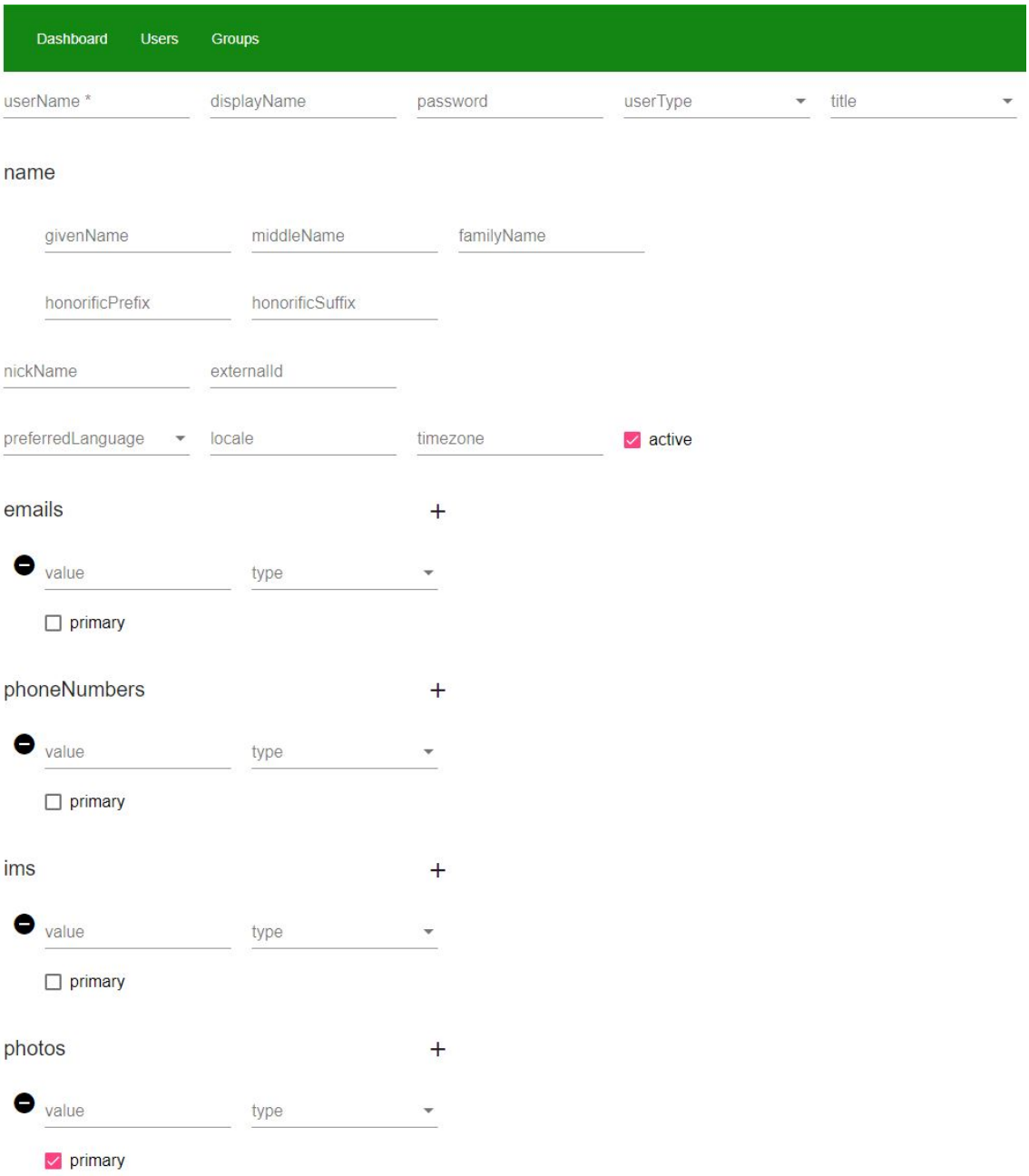


Figure 5.11: Add user page

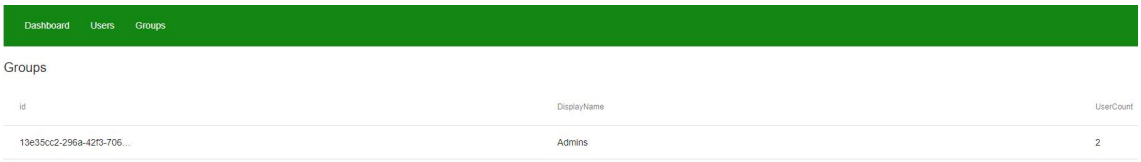


Figure 5.12: GroupList

5.3.2 Group Pages

The group pages have the same concept as the user pages, meaning the three views are list-, full-, and add group. The list page shows the user the id, display name, and user count of the groups, figure 5.12 only has one group and the group has 2 users in it. When the user clicks the group they want to open the user is redirected to the full group page. The full group page retrieves the group data from the back-end using the id and then renders the page view with the data received.

Figure 5.13 illustrates the view using test data. The data that is received from the back-end is in SCIM format and the data is shown in that way in the UI. The reason why the view has add users and edit group buttons at the top is that during development I was having problems adding users to groups because the PATCH method was not implemented in the back-end. Therefore the add users button sends a PUT method to the backend "/adduser" endpoint to add users and the edit group sends a PUT method to the default /groups/id endpoint in the back-end. The edit group button allows users to change the display name of the group and delete users from the group. The changes made on the page must be saved using the save changes button that invokes the PUT method. The same idea is behind adding users, but the add users button opens a list of users that can be added to the group. The user selects the wanted users from the list and then clicks the add selected users button to send the HTTP method to the back-end.

The final page used in groups is the add group page which can be seen in figure 5.14. The add group page uses the user list that is also used in the add users in the full group page, this was done because the dynamic form can crate the list without having to build

my lists. The user can add a display name and then select the users from the list. When the selections are done the save button will invoke the HTTP POST method that takes the data from the list and display name to send a SCIM style JSON message to the back-end.

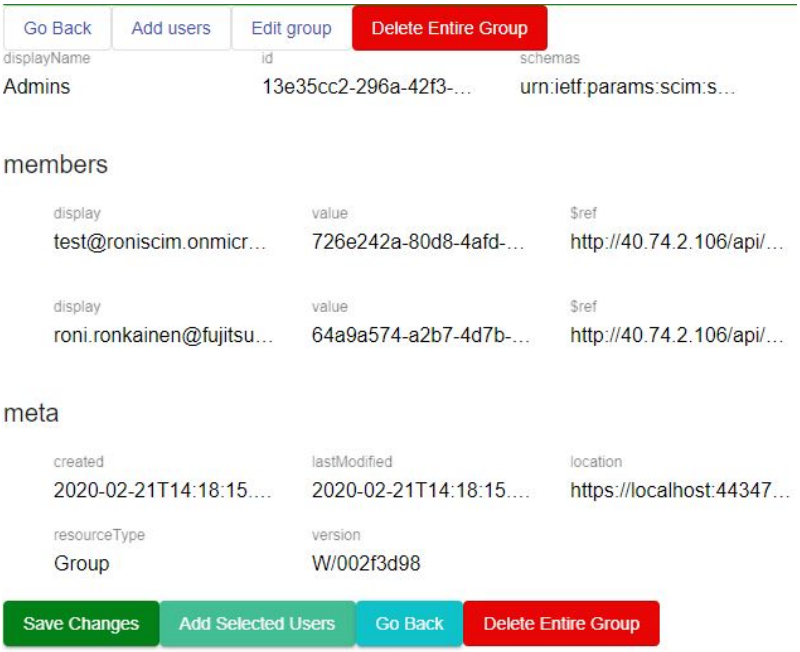


Figure 5.13: Full group page

Figure 5.14: Add group page

5.3.3 Improvements

The UI was made using Angular so that I will learn more about the framework, and also because I could use already existing components especially the dynamic form component in my project. The use of dynamic form created some problems in keeping the data in pure SCIM format, but then again it is not necessary for the UI. Improvements to the UI would be to make it more user friendly by creating styles that are adjusting the layout. Other improvements would be to make the group modification to be done with a PATCH operation because the back-end now supports it. There are endless possibilities to improve the system of course but for a testing tool, the main idea is so that the user can view and modify data.

5.4 Testing

This section will cover the testing phase of the project. The testing was continuous throughout the building process of the application to make sure all the HTTP methods would work properly. This was primarily done by hand using the Postman [27] application to send data to the back-end with JSON examples taken from the SCIM core schema [2] documentation. That kind of testing would not be very beneficial to show and therefore it is only acknowledged. The real testing started from first getting the information flow to go from source UI -> source back-end -> target back-end -> target UI, this is the first part of what tested if the application could handle the SCIM messages. The second phase was to test that could my application be run in a Docker container using an azure database and be provisioned by Azure Active Directory. If the application could handle all that then it could be accepted as a real testing tool.

The first part of the testing was quite easy to set up because I had already tested that the application would work separately with the HTTP methods. I just configured that the source would send the HTTP messages, related to data handling forward to the target

application as well when the source UI was used. The two systems started exchanging data with each other as expected and the changes made in the source UI would show up in the target. There were several bugs that I noticed during this testing, the biggest one was that when creating a new user the external id of the target was not the id of the source resource. This meant that the target back-end needed to be changed to fix this problem.

Other minor bugs were found and they were fixed for example unique name checks in the target and wrong version numbering in meta of users were found and fixed. The testing was only done by me and did not involve any automated tests. During this testing phase, I decided that it was not worth the effort of writing automated tests for these situations. During this testing phase, I got the necessary SCIM HTTP methods working, but the PATCH operation was still missing. This was not a problem in this situation as I had the SCIM style JSON messages going through the whole system from source to target. The fact that everything was working as expected I decided to move to the next phase.

The phase two of testing was to provide a real working connection with a trusted certified service provider. Azure Active Directory was chosen to be the service provider because it provided an easy to use implementation. I used the instructions provided by Microsoft [28] on how the connection should be done. This was when I realized that I would have to make some changes to my project at hand. The biggest thing was to create the PATCH method and add the query options. After the implementation of these two things, I could start testing my application in a real situation.

To start I had some help to get my target application running in a Docker container using an external database. The database and Docker container is provided by Fujitsu and they are run in Azure. The real task was to create another azure account and have it run an enterprise application that could provide the target application. Figure 5.15 shows how the test connection worked with the provided credentials and figure 5.16 shows how the provisioning worked succeeded in the target application. The connection test sends the target application a query message using a random id and expects the target to respond

with an empty list response. The query implementations in the target application support the query using the id and that is why the connection test was successful. The provisioning was done automatically to the target after the test connection was successful.

In real applications that would use real identities, the system would need an OAuth bearer token, but in this implementation, it is not necessary and therefore is left out of the testing. Azure saved logs about the synchronization process and they give more information about the provisioning, see figure 5.17. The logs showed that my project target was not completely compliant with the Azure implementation because I did not have the URI's required in all the resources. The main point is that all the groups are sent into the database and the figures 5.14 and 5.11 in the UI section shows how the provisioning worked.

The testing phase of this project was kept short and it was only to show that the project works with another SCIM build and that I could see what else should be done in the project. The outcome of the test was a success and it provided the necessary verification that the SCIM application can work with another application that uses the SCIM protocol style message formats.

5.5 Observations

During the process of making the project and writing this thesis, there have been times where I have noticed some aspects to consider that are not groundbreaking but make life a bit harder in more complex application architectures. The biggest thing that is not taken into account in the specification is the asynchronous actions that most identity management tools have. The SCIM specification states that the receiving party must respond to the source system with an HTTP message saying the process is done. The problem with this is that in most cases the responding service will need to first create the user and this might take time. If the service provider is not configured to continue its

process even without the HTTP response, the service provider will work slow and create a bottleneck if there are multiple tenants.

The decision of not taking into account the synchronous or asynchronous workflows in the specification is not completely a bad thing. The SCIM specification is not meant to be a guide on how the system should be built because it gives the builder free hands on the architecture. When there is only one tenant in the service provider it is not necessary to have asynchronous workflows. The service provider does not need to change data in multiple places at a time meaning the work can be done in a waterfall process. The real problem is when the service provider has multiple tenants and it sends out multiple HTTP calls at the same time. This will block the processes and make the service slow, the solution to this problem is using asynchronous data transfer where the service provider will continue working on other processes enabling the service provider to deploy more HTTP calls.

The only other part that I would consider a worry is that SCIM gives quite a lot of freedom for the resources. This does open up the possibility to improve systems for specific clients, but will always mean that the work will not be standardized. The problem here is that SCIM is supposed to create a standard way of sending and receiving known objects so that it could just be attached to a working solution. This is not the case if all the tenants require modifications to their resources. This is more of a problem if SCIM is marketed as a custom solution to all clients.

Looking at the results of the project and the thesis it can be seen that the SCIM specification creates a good basis on where to start building. With this I mean it does not matter what programming language or database is used in creating the applications that use SCIM. The only things that the SCIM specifications limit is the resources and that they must be sent with HTTP calls in JSON format. This itself is the great part of SCIM where a web application that uses for example Azure [4] to host the application, can exchange data with a console application running on a Linux machine.

The SCIM website `simplecloud.info` [1] contains a list of SCIM applications that use a variety of different coding languages. This is a testament to the versatility that SCIM brings. So how do my findings relate to every other SCIM application? For example, the problem of synchronous and asynchronous responses affect all the SCIM applications, but specific challenges like not having the right join tables in C# should not be considered. Most problems encountered in the project e.g. creating the PATCH call can be viewed as universal problems. In this case, the problem is that the PATCH call is very complex to build since it has so many variations. The easiest solution is to create switch cases that go through the whole database. This method will need tons of loops increasing the complexity of the code and slowing down performance. The problem in itself is not dependant on the coding language.

Even though the project is coded with C# and the front-end is Angular the basic resources from my database could be used to create another SCIM application using completely different coding languages. This means that most of the findings in this thesis are universal and can be applied to other cases. Excluding the front-end in the project, because the SCIM specification does not specify the need for a UI. This is one reason why I think that SCIM will be even more common in the future.

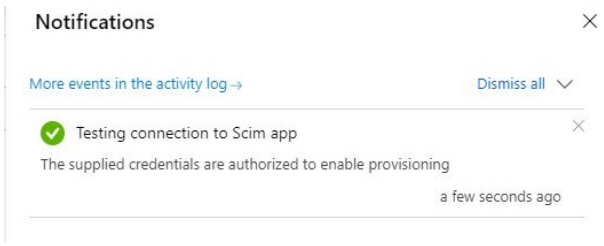


Figure 5.15: Test connection

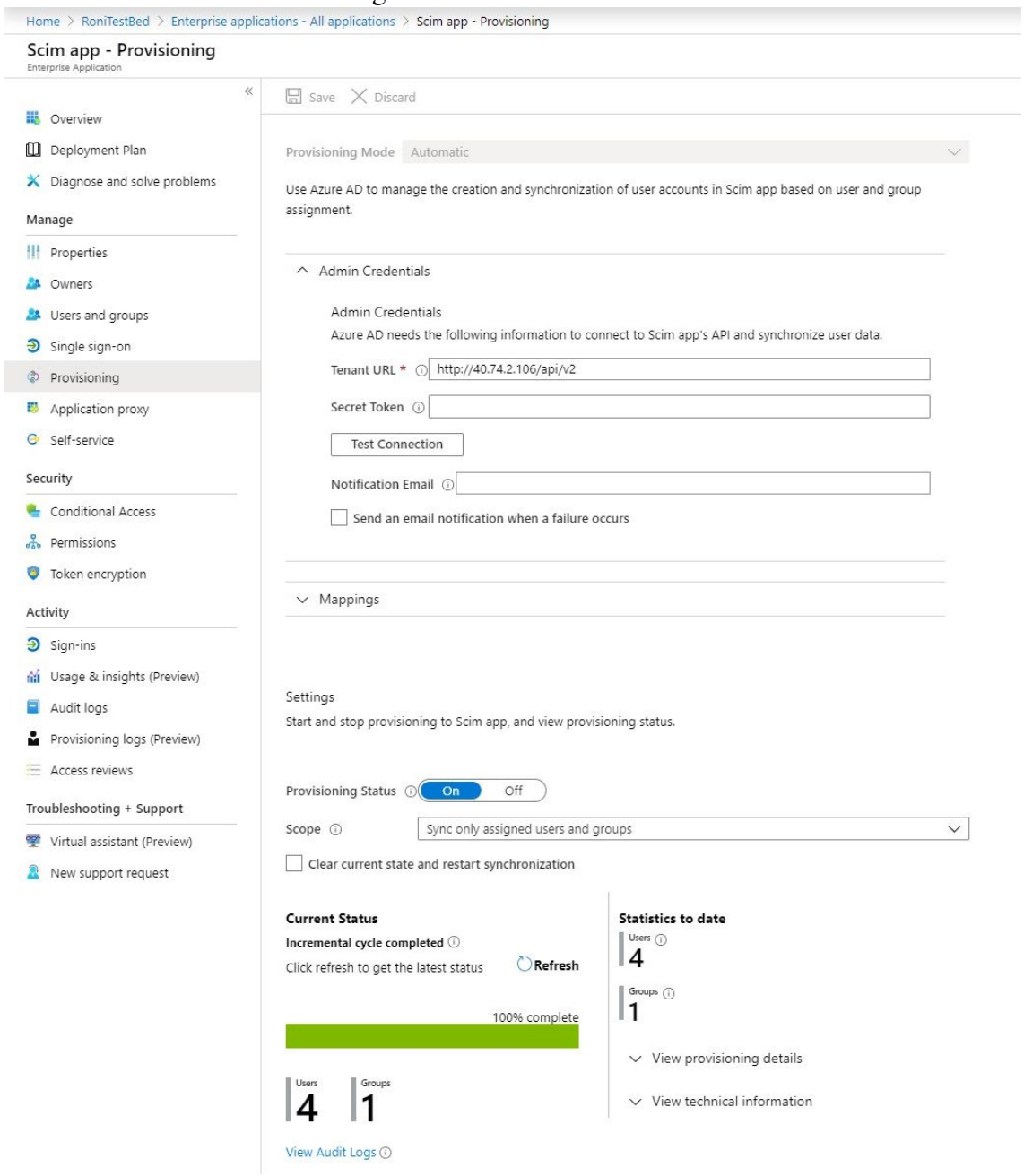


Figure 5.16: Provisioned

Date	Service	Category	Activity
2/21/2020, 2:18:15 PM	Account Provisioning	ApplicationManagement	Export
2/21/2020, 2:18:15 PM	Account Provisioning	ApplicationManagement	Export
2/21/2020, 2:18:15 PM	Account Provisioning	ApplicationManagement	Import
2/21/2020, 2:18:15 PM	Account Provisioning	ApplicationManagement	Other
2/21/2020, 2:18:15 PM	Account Provisioning	ApplicationManagement	Other
2/21/2020, 2:18:15 PM	Account Provisioning	ApplicationManagement	Export
2/21/2020, 2:18:15 PM	Account Provisioning	ApplicationManagement	Synchronization rule action
2/21/2020, 2:18:15 PM	Account Provisioning	ApplicationManagement	Import
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Import
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Export
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Export
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Export
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Synchronization rule action
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Synchronization rule action
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Synchronization rule action
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Import
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Import
2/21/2020, 2:18:14 PM	Account Provisioning	ApplicationManagement	Import
2/21/2020, 2:18:13 PM	Account Provisioning	ApplicationManagement	Import
2/21/2020, 2:18:13 PM	Account Provisioning	ApplicationManagement	Import
2/21/2020, 2:18:13 PM	Account Provisioning	ApplicationManagement	Import
2/21/2020, 2:18:11 PM	Account Provisioning	ApplicationManagement	Other
2/21/2020, 2:18:11 PM	Account Provisioning	ApplicationManagement	Other

Status	Status reason	Target(s)	Initiated by (actor)
Success	Group 'Admins' was updated in customappsso	Scim app, Admins	Azure AD Cloud Sync
Success	Group 'Admins' was updated in customappsso	Scim app, Admins	Azure AD Cloud Sync
Success	Retrieved 'Admins' from customappsso	Scim app, Admins	Azure AD Cloud Sync
Success	Retrieved 'Admins' from customappsso. The sourc...	Scim app, Admins	Azure AD Cloud Sync
Success	Retrieved 'Admins' from customappsso. The sourc...	Scim app, Admins	Azure AD Cloud Sync
Success	Group 'Admins' was created in customappsso	Scim app, Admins	Azure AD Cloud Sync
Success	Group 'Admins' will be created in customappsso (...	Scim app, Admins	Azure AD Cloud Sync
Success	No urn:ietf:params:scim:schemas:core:2.0:Group wi...	Scim app, Admins	Azure AD Cloud Sync
Success	Received Group 'Admins' change of type (Add) fro...	Scim app, Admins	Azure AD Cloud Sync
Success	User 'Test3@roniscim.onmicrosoft.com' was create...	Scim app, Test3@roniscim.onmicrosoft.com	Azure AD Cloud Sync
Success	User 'test@roniscim.onmicrosoft.com' was created...	Scim app, test@roniscim.onmicrosoft.com	Azure AD Cloud Sync
Success	User 'roni.ronkainen@fujitsu.com' was created in c...	Scim app, roni.ronkainen@fujitsu.com	Azure AD Cloud Sync
Success	User 'test@roniscim.onmicrosoft.com' will be creat...	Scim app, test@roniscim.onmicrosoft.com	Azure AD Cloud Sync
Success	User 'Test3@roniscim.onmicrosoft.com' will be cre...	Scim app, Test3@roniscim.onmicrosoft.com	Azure AD Cloud Sync
Success	User 'roni.ronkainen@fujitsu.com' will be created i...	Scim app, roni.ronkainen@fujitsu.com	Azure AD Cloud Sync
Success	No urn:ietf:params:scim:schemas:extension:enterpr...	Scim app, Test3@roniscim.onmicrosoft.com	Azure AD Cloud Sync
Success	No urn:ietf:params:scim:schemas:extension:enterpr...	Scim app, roni.ronkainen@fujitsu.com	Azure AD Cloud Sync
Success	No urn:ietf:params:scim:schemas:extension:enterpr...	Scim app, test@roniscim.onmicrosoft.com	Azure AD Cloud Sync
Success	Received User 'Test3@roniscim.onmicrosoft.com' c...	Scim app, Test3@roniscim.onmicrosoft.com	Azure AD Cloud Sync
Success	Received User 'roni.ronkainen@fujitsu.com' chang...	Scim app, roni.ronkainen@fujitsu.com	Azure AD Cloud Sync
Success	Received User 'test@roniscim.onmicrosoft.com' ch...	Scim app, test@roniscim.onmicrosoft.com	Azure AD Cloud Sync
Success	Due to configuration changes, we need to to re-sy...	Scim app	Azure AD Cloud Sync
Success	Due to configuration changes, we need to to re-sy...	Scim app	Azure AD Cloud Sync

Figure 5.17: Provisioning logs

Chapter 6

Conclusion

The thesis explains what the SCIM specification is and creates an application using the specification. Identity and access management is briefly explained to better understand why SCIM was made and what it can solve. SCIM was created to make an easy to use specification that handles identities in cross-domain situations well. The implementation part of the thesis was to build a test tool for Fujitsu to test that their identity management application can handle SCIM messages. Implementation was tested in two phases to ensure that it was built properly and the messaging happened in SCIM format.

The thesis was meant to provide the reader with an understandable view of the SCIM specifications by showcasing the Core Schema [2] and Protocol [3]. The other task was to build a testbed that could test if other applications are SCIM compatible. The first problem of creating an understandable overview of the SCIM specification was done in chapters three and four. The testbed meets the requirements of a working SCIM application. The verification for success was achieved by connecting the testbed to Microsoft Azure Active Directory [4] for provisioning.

The research focused mostly on SCIM, but IAM was also a part of the work. The reason for discussing IAM is that SCIM is heavily linked to it. IAM is growing in popularity and the need for working applications is increasing. In this time of age not only big companies use IAM applications, but also smaller companies can gain from the ser-

vices. Businesses have realized that providing software as a service is very beneficial and cost-effective. That is why SCIM and other cloud-based identity management specifications are so important. SCIM is used by big technology companies and that is why it was chosen as the specification in this thesis. The core schema provides the backbone of the specification by defining what type of data should be handled in SCIM. The protocol specifies how the applications using SCIM should handle the data sent. These two documents work in tandem to provide a working specification that handles identities very efficiently.

The project that was done for this thesis is the actual product that Fujistu wanted. The documentation and research are there to back up the two applications and served as a learning opportunity for the author. The project was done from scratch because the already existing projects done for SCIM did not meet the requirements of the test tool. The deciding factors were that the project should be written in C# and Angular, so that in the future the tool could be modified by the IDM team, also so that I could enhance my skills in those two languages. The outcome of the project was a success and the two testing phases verified that the applications work. The thing that I learned from working with SCIM is that the format of data sent in the JSON files can sometimes be very hard to implement in the actual code.

SCIM has gained popularity and big companies have started using it in their IDM applications, this means that more companies will be following the example and it seems that SCIM will become the norm in handling identity data. The main reason in my mind is because of the simplicity that the SCIM specification has. The implementation does not require lots of time just a working API that can retrieve data from a service provider. Services and applications are all going to the cloud and this means there is a need for a single accepted specification which SCIM could be. As of now, it seems SCIM will be the most used specification.

References

[1] SCIM: System for Cross-domain Identity Management. Online, available at http://www.simplecloud.info/. last accessed: 12.06.2020.

[2] Grizzle K. Wahlstroem E. Hunt, P and C. Mortimore. System for cross-domain identity management: Core schema. Technical report, RFC 7643, DOI 10.17487/RFC7643, September 2015, online available at: https://tools.ietf.org/html/rfc7643.

[3] Grizzle K. Ansari M. Wahlstroem E. Hunt, P. and C. Mortimore. System for cross-domain identity management: Protocol. Technical report, RFC 7644, DOI 10.17487/RFC7644, September 2015, online available at: https://tools.ietf.org/html/rfc7644.

[4] Microsoft Azure. Azure Active Directory. Online, available at https://azure.microsoft.com/en-us/services/active-directory/. last accessed: 12.06.2020.

[5] Flexera. State of the Cloud Report 2019. Online, available at https://resources.flexera.com/web/media/documents/rightscale-2019-state-of-the-cloud-report-from-flexera.pdf?elqTrackId=372b6798c7294392833def6ec8f62c5c&elqaid=4588&elqat=2&_ga=2.75255952.381106268.1556868633-1445624021.1556868633. last accessed: 05.06.2020.

[6] TechTarget. What is Identity and Access Management? Online, available at https://searchsecurity.techtarget.com/definition/identity-access-management-IAM-system, May 2019. last accessed: 12.06.2020.

[7] CSO Online. What is IAM? Identity and access management explained. Online, available at https://www.csoonline.com/article/2120384/what-is-iam-identity-and-access-management-explained.html, Oct 2018. last accessed: 12.06.2020.

[8] Tim Mather, Subra Kumaraswamy, and Shahed Latif. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. O'Reilly Media, Inc., 2009.

[9] Sameera Abdulrahman Almulla and Chan Yeob Yeun. Cloud Computing Security Management. In *2010 Second International Conference on Engineering System Management and Applications*, pages 1–7. IEEE, 2010.

[10] Azure AD | Microsoft Docs. Automated saas app user provisioning in azure ad. Online, available at https://docs.microsoft.com/en-us/azure/active-directory/app-provisioning/user-provisioning, Nov 2019. last accessed: 11.06.2020.

[11] Alex Chi Keung (ed.) Ng. *Contemporary identity and access management architectures : emerging research and opportunities*. IGI Global, Hershey, Pennsylvania (701 E. Chocolate Avenue, Hershey, Pennsylvania, 17033, USA), 2018.

[12] John Vacca. *Computer and information security handbook, Third edition. 2017*. Morgan Kaufmann Publishers, Cambridge, MA.

[13] Heinz Johner, Larry Brown, Franz-Stefan Hinner, Wolfgang Reis, and Johan Westman. *Understanding LDAP*, volume 6. IBM, 1998.

[14] B Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications magazine*, 32(9):33–38, 1994.

[15] GlobalSign. Ssl/tls certificate validity is now capped at a maximum of two years. Online, available at `https://www.globalsign.com/en/blog/ssl-certificate-validity-capped-at-maximum-two-years`. last accessed: 12.06.2020.

[16] David Cooper, Stefan Santesson, S Farrell, Sharon Boeyen, Rusell Housley, and W Polk. RFC 5280: Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile. *IETF, May*, 2008.

[17] Jenny Torres, Michele Nogueira, and Guy Pujolle. A Survey on Identity Management for the Future Network. *IEEE Communications Surveys & Tutorials*, 15(2):787–802, 2012.

[18] Nat Sakimura, John Bradley, Mike Jones, Breno De Medeiros, and Chuck Mortimore. Openid connect core 1.0 incorporating errata set 1. *The OpenID Foundation, specification*, 335, 2014.

[19] Roy Fielding and Julian Reschke. Hypertext transfer protocol (http/1.1): Semantics and content. Online, available at `https://tools.ietf.org/html/rfc7231`, 2014.

[20] E Lear and P Eggert. Procedures for Maintaining the Time Zone Database. Technical report, RFC 6557, accessed at `https://tools.ietf.org/html/rfc6557`, 2012.

[21] D Crocker and P Overell. Augmented BNF for Syntax Specifications: ABNF. Technical report, RFC 5234 (obsoletes RFC 2234, RFC4234), accessed at `https://tools.ietf.org/html/rfc5234`, 2008.

[22] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing*, PP:1–1, 06 2018.

[23] Microsoft Docs. C# docs - get started, tutorials, reference. Online, available at `https://docs.microsoft.com/en-us/dotnet/csharp/`. last accessed: 12.06.2020.

[24] Google. Angular. Online, available at `https://angular.io/`. last accessed: 12.06.2020.

[25] Microsoft SQL Server. Apply intelligence across all your data with sql server 2019. Online, available at `https://www.microsoft.com/en-us/sql-server/sql-server-2019`. last accessed: 12.06.2020.

[26] Microsoft Docs. .NET Core documentation. Online, available at `https://docs.microsoft.com/fi-fi/dotnet/core/`. last accessed: 12.06.2020.

[27] Postman. The Collaboration Platform for API Development. Online, available at `https://www.postman.com/`. last accessed: 12.06.2020.

[28] Microsoft Docs. Develop a SCIM endpoint for user provisioning to apps from Azure AD. Online, available at `https://docs.microsoft.com/en-us/azure/active-directory/app-provisioning/use-scim-to-provision-users-and-groups`. last accessed: 12.06.2020.