

Decentralized Solution to Mitigate Job Posting Scams by Proving Ownership and Identity

Configuration Manual

MSc Research Project
Cloud Computing

Apostolos Giannakidis
Student ID: x20124066

School of Computing
National College of Ireland

Supervisor: Horacio Gonzalez-Velez

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Apostolos Giannakidis
Student ID:	x20124066
Programme:	Cloud Computing
Year:	2021-22
Module:	MSc Research Project
Supervisor:	Horacio Gonzalez-Velez
Submission Due Date:	15/Aug/2022
Project Title:	Decentralized Solution to Mitigate Job Posting Scams by Proving Ownership and Identity - Configuration Manual
Word Count:	10291
Page Count:	49

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	<i>Apostolos Giannakidis</i>
Date:	15/Aug/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Decentralized Solution to Mitigate Job Posting Scams by Proving Ownership and Identity Configuration Manual

Apostolos Giannakidis
x20124066

Contents

1	Introduction	1
2	Project Links	1
3	Prerequisites	2
4	Cloud Setup	4
4.1	Azure Cloud	4
4.1.1	Setting up the fake company website	4
4.1.2	Configuring the Azure Verifiable Credentials service	6
4.1.3	Setting up the Verifiable Credentials Issuer	14
4.2	Firebase Cloud	15
4.3	Pinata Cloud	17
4.4	Alchemy Web3	17
4.5	NoCode API for screenshot generation	18
5	LinkedIn Identity Federation Setup	20
6	NFT Deployment on Polygon	21
7	AdvertChain Software	25
7.1	Deploying the AdvertChain backend	26
7.2	Deploying the AdvertChain frontend	29
8	Running a Demo of the Proof-of-Concept	32
8.1	Prerequisites	32
8.2	User Stories	32
8.3	The job poster user journeys	33
8.3.1	Issuing an employee Verifiable Credential to the job poster	33
8.3.2	Minting job posts as <i>AdvertChain</i> NFTs	34
8.4	The job applicant user journey	42
8.5	Browsing the <i>AdvertChain</i> NFTs	44
9	Frequently Asked Questions	45

1 Introduction

AdvertChain is the Proof-of-Concept system of this research paper. It is a Cloud-based, decentralized, solution that aims to protect job applicants against the online job scam. After the system verifies the identity of the job poster and the hiring company using DIDs and VCs, it authorizes the job poster to mint a LinkedIn job post as NFT. Then, job applicants can make use of the provided REST API endpoint to validate the ownership of LinkedIn job posts and the identity of job posters.

Due to its decentralized design and its Cloud-based architecture, the system consists of several components. This configuration manual is provided to assist with setting up all the components needed to run the *AdvertChain* research project from scratch. If you already have performed all the prerequisite steps and you poses a Verifiable Credential issued by a verified company and you only wish to use *AdvertChain* and start minting job posts as new NFTs, then go to [section 8](#).

2 Project Links

The following links provide all the resources required to deploy and run the *AdvertChain* Proof-of-Concept project from scratch. Some of these links, such as the domain of the fake company have expiration dates. Thus, some of the links will stop working after a year or so. They are provided for completeness of the MSc report and to assist the college reviewers for assessing this project.

The source code of the implementation of the *AdvertChain* Proof-of-Concept project can be found in this GitHub repo: <https://github.com/maestros/advertchain-portal>.

The source code of the implementation of the fake company's website can be found in this GitHub repo: <https://github.com/maestros/grecho-website>. Also, the source code for the VC issuer of the *Grecho* fake company can be found in this GitHub repo: <https://github.com/maestros/active-directory-verifiable-credentials-node>.

For the purposes of an end-to-end demo we need a company that will post job posts on LinkedIn and will use *AdvertChain* to protect the ownership and identity of these job posts. For this reason, a fictitious company was created. The name of this fake company is *Grecho* and the company's website can be accessed via: <https://www.grecho.site>. The VC issuing service of *Grecho* can be accessed via: <https://grecho-vc-issuer.azurewebsites.net>. The LinkedIn company page of this fake company is: <https://www.linkedin.com/company/grecho/>. *Grecho* requires a LinkedIn company page in order to create fake job posts to demonstrate the NFT minting functionality and the job post validation functionality.

In order to verify *Grecho*'s domain to the Azure Verifiable Credentials service, a JSON file had to be deployed on *Grecho*'s website. This JSON file is accessible via: <https://www.grecho.site/.well-known/did-configuration.json>. This file contains the DID of *Grecho*. Effectively, this file allows the Azure Verifiable Credentials service to issue VCs to *Grecho*'s employees with verified domain. This way, VC verifiers can be confident that the presented VCs are issued by the issuer *Grecho* that has a verified domain.

The *AdvertChain* portal that allows users to authenticate and be authorized to mint new NFTs is accessible via: <https://advertchain-demo.web.app/>. The LinkedIn company

page of *AdvertChain* is: <https://www.linkedin.com/company/advertchain/>. *AdvertChain* requires a LinkedIn company page in order to enable Identity Federation.

The *AdvertChain* job post validator is accessible via the REST API endpoint <https://us-central1-advertchain-firebase.cloudfunctions.net/api/validateJobPost> that accepts the URL of the LinkedIn job post via the *jobUrl* GET query parameter.

The *AdvertChain* ERC-721 Smart Contract can be accessed via Polygonscan: <https://mumbai.polygonscan.com/address/0xC3a2eb67D1d0Df8bc49AFf5aA4EcB540Ca4A7D5F#code>. The *AdvertChain* ERC-721 Smart Contract is already verified. Thus, its source code is also accessible via Polygonscan.

The *AdvertChain* NFTs that have been minted for demo purposes can be viewed via OpenSea: <https://testnets.opensea.io/collection/advertchain>. Note that *AdvertChain* NFTs cannot be traded. We use OpenSea as a convenience method only to view the set of minted NFTs. Even if a user tries to sell, buy or transfer the ownership of an *AdvertChain* NFT, the transaction will fail due to the way the Smart Contract has been designed.

3 Prerequisites

To better understand how this solution works and how to properly set it up, some understanding of the following concepts and technologies is needed:

1. Cloud Computing (PaaS, Serverless)
2. Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs)
3. Solidity, Ethereum and Non-Fungible Tokens (NFTs)
4. InterPlanetary File System (IPFS)
5. Identity Federation
6. Public Key Infrastructure (PKI)
7. React.js and Node.js

To run this solution in an end-to-end fashion, the following prerequisites are required:

1. An Azure Cloud account
2. A Google Cloud Firebase account
3. A Pinata Cloud account
4. A NotCode API account
5. A LinkedIn account
6. An Alchemy Web3 account
7. An Polygonscan account
8. A MetaMask wallet with test Matic tokens
9. The Microsoft Authenticator mobile app

Following are high-level instructions on how to set up the above-mentioned prerequisites:

1. To setup an Azure account, visit the page: <https://azure.microsoft.com/en-us/free/>. Microsoft provides many services that are always free, some services that are free for 12 months as well as \$200 in Azure credit for the first month. For our Proof-of-Concept use case, this Cloud offering is more than enough. Note that for the creation of this account some payment details will have to be entered. However, no cost is expected to be charged in your account for the purposes of this PoC.
2. To create a Google Cloud Firebase account, visit the page: <https://firebase.google.com/>. Once you login to your new Firebase account, create a new project and upgrade it to the Blaze Pay as you go billing plan. This plan is required, however, no cost is expected to be charged in your account for the purposes of this Proof-of-Concept. If you wish to be sure that no cost will be incurred, create a budget for 1 USD.
3. To create a Pinata Cloud account, visit the page: <https://www.pinata.cloud/>. The free tier account works fine for this Proof-of-Concept.
4. To create a NoCode API account, visit the page: <https://nocodeapi.com/>. The free tier account works fine for this Proof-of-Concept.
5. If you do not have a personal LinkedIn account, visit the page: <https://www.linkedin.com/signup/>. The free tier account works fine for this PoC.
6. To create an Alchemy Web3 account, visit the page: <https://auth.alchemyapi.io/signup>. The free tier account works fine for this Proof-of-Concept.
7. To create a Polygonscan account, visit the page: <https://polygonscan.com/register>. The free tier account works fine for this Proof-of-Concept.
8. To use MetaMask, first install the browser plugin from your browser's plugin page or from the page: <https://metamask.io/download/>. Select the Mumbai network and in the the MetaMask settings change the Mumbai network's RPC URL to: **<https://rpc-mumbai.maticvigil.com>**. Next, create a wallet, select the Mumbai testnet and copy the wallet's address to a notepad. Finally, add test Matic to your wallet via: <https://faucet.polygon.technology> and paste your wallet's address.
9. To install the Microsoft Authenticator app on your mobile phone, visit the page: <https://www.microsoft.com/en-us/security/mobile-authenticator-app>.

If you do not wish to deploy *AdvertChain* from scratch and you only wish to mint new NFTs, then you need to make sure you have the following:

1. You have a personal LinkedIn profile
2. You have a Verifiable Credential issued by the hiring company
3. The hiring company has posted a job post on LinkedIn
4. You have installed MetaMask and you have test Matic in your Mumbai wallet.

If all the above prerequisites are done, then you can proceed to [section 8](#) for instructions on how to start minting new *AdvertChain* NFTs.

4 Cloud Setup

4.1 Azure Cloud

For the end-to-end demo of this solution, we use Microsoft’s Azure Cloud for two (2) main use cases:

1. Verifiable Credentials (VC) issuance and verification service
2. Website and VC issuer of fake company

For simplicity of the Proof-of-Concept implementation, we will use the same Azure subscription and tenant to configure and deploy the Cloud services for both use cases. In a real-world scenario, the company’s website and VC issuer would be deployed in separate Cloud environments and tenants because these two are unrelated and completely different entities. Remember that the fake company, *Grecho*, represents a company that posts job posts on LinkedIn and wants the job posts to be verifiable by *AdvertChain*.

4.1.1 Setting up the fake company website

We will start the process by setting up the fake company website. We need this website before we proceed to the configuration of the Azure Verifiable Credentials service.

For the registration of the domain, any domain registrar could have been used. In our case, we used GoDaddy, one of the most popular domain registrars.

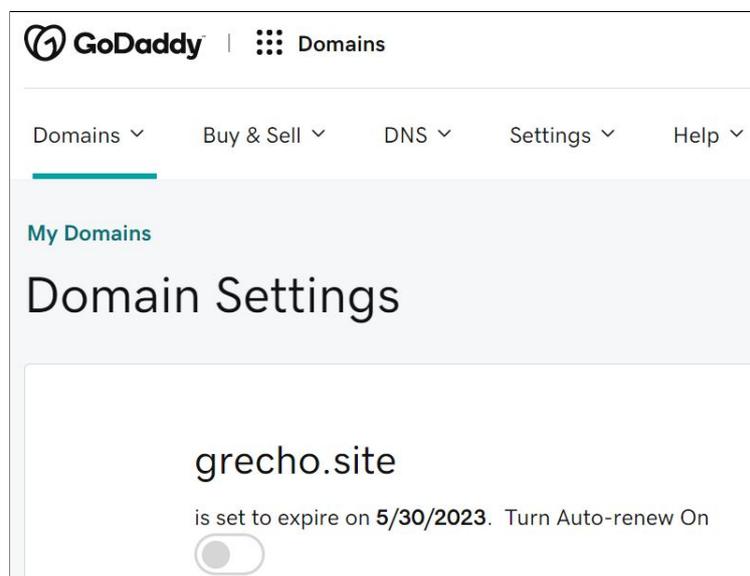


Figure 1: Domain registration on GoDaddy for *Grecho*

After registering a domain, we must ensure to change its DNS servers to point to the Azure Static Web Apps service, which is the service used to host the *Grecho* website. Figure 2 shows the Azure nameservers that we used in our PoC.

Next, we must create the website’s content. For the purpose of this PoC we only need a static website. The source code of this static website is accessible on this GitHub repo: <https://github.com/maestros/grecho-website>. The GitHub repo has been configured with a GitHub action that builds the project when a new PR is merged and deploys the new artifact on Azure Static Web Apps. This way we achieve Continuous Deployment. The

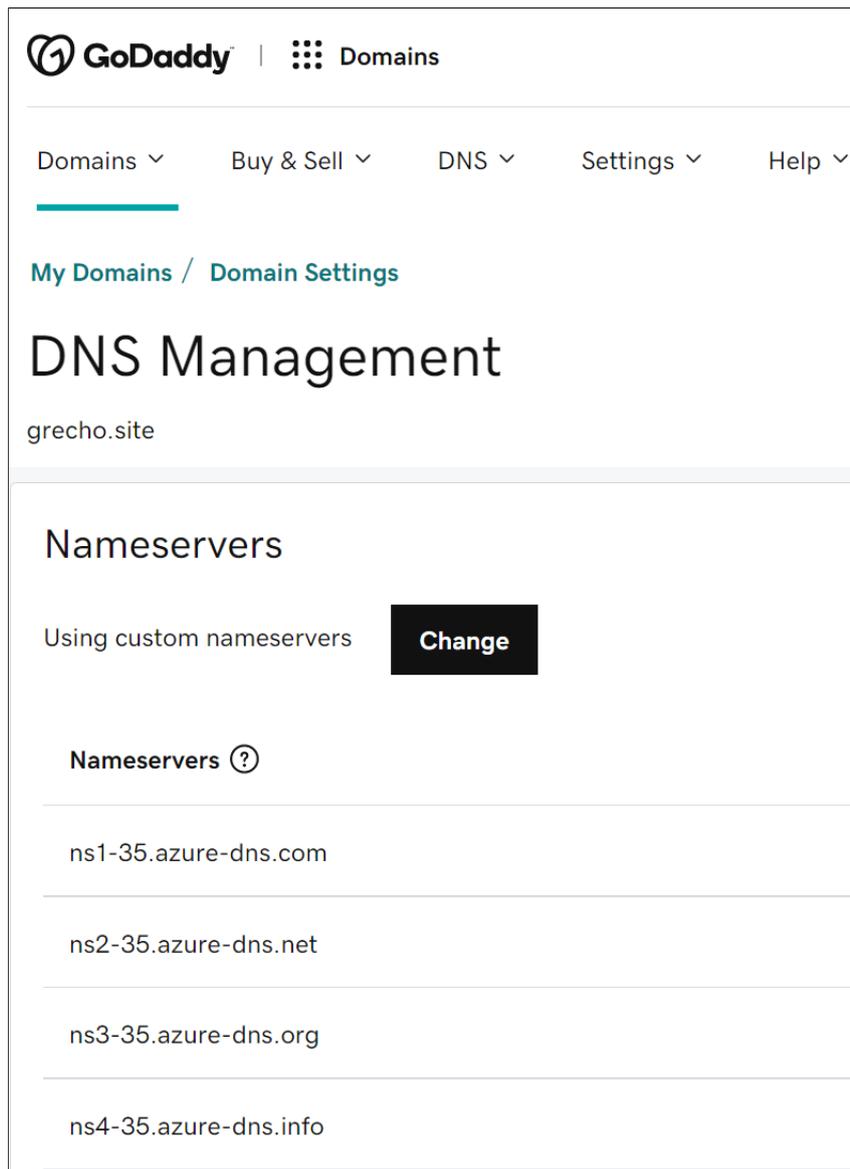


Figure 2: Nameserver configuration for *Grecho*

figure 3 shows the GitHub Action that is configured for this repo. On the Azure side, we must create a new Static Web App with GitHub as a source and the main branch of the above repo as the deployment branch. The figure 4 shows the configuration of the Static Web App on Azure for *Grecho*. Finally, to add a validated custom domain for the Static Web App, we added the custom domain <http://www.grecho.site> in the Custom Domains section and we validated the domain by adding the necessary CNAME in the DNS zone, as shown in the screenshot at Figure 5.

Apart from the static webpage of the *Grecho* website, there is one more important webiste page that we need. The page that issues Verifiable Credentials to the *Grecho* employees. However, before we proceed to the creation of this page, we must first set up the Azure AD Verifiable Credentials service.

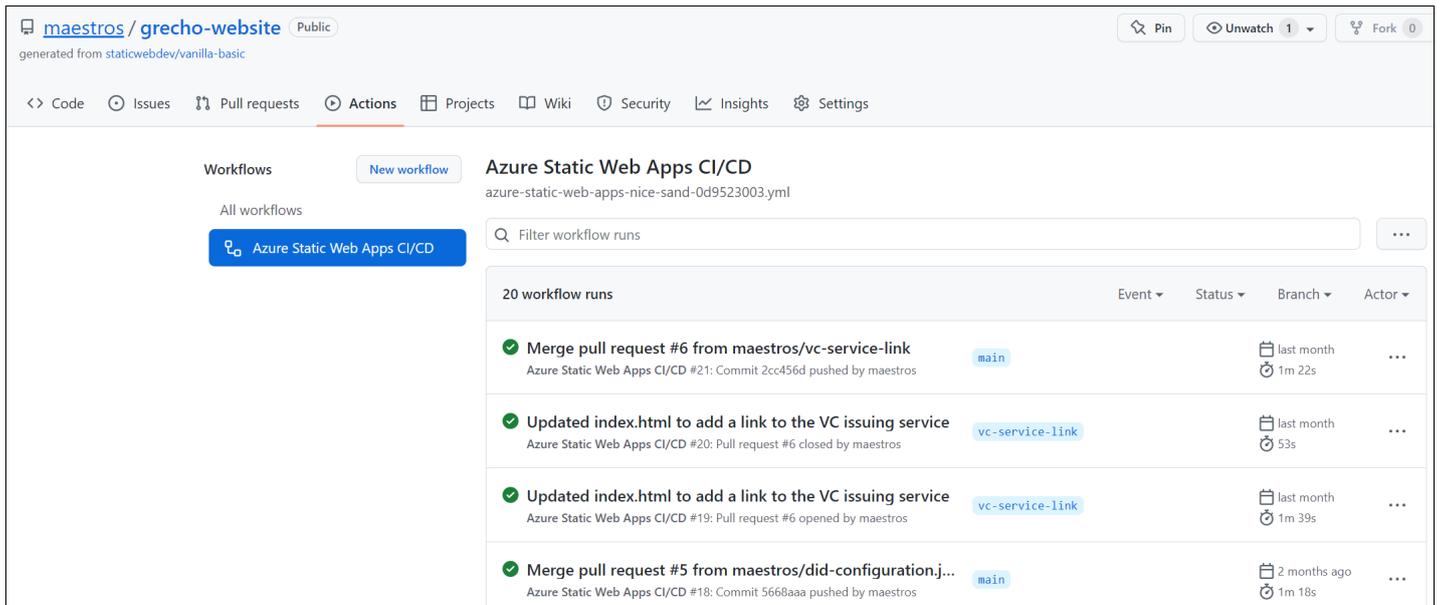


Figure 3: *Grecho* website GitHub Action for Continuous Deployment

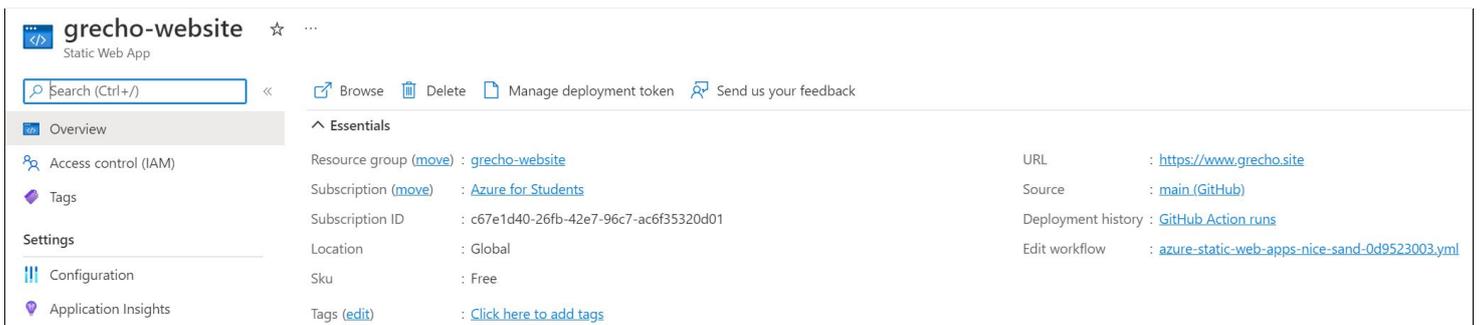


Figure 4: *Grecho* Static Website App setup on Azure

4.1.2 Configuring the Azure Verifiable Credentials service

The Azure Verifiable Credentials service was recently launched as a General Availability service under the name Microsoft Entra Verified ID. Microsoft's [public documentation](#) provides detailed instructions how to configure the service.

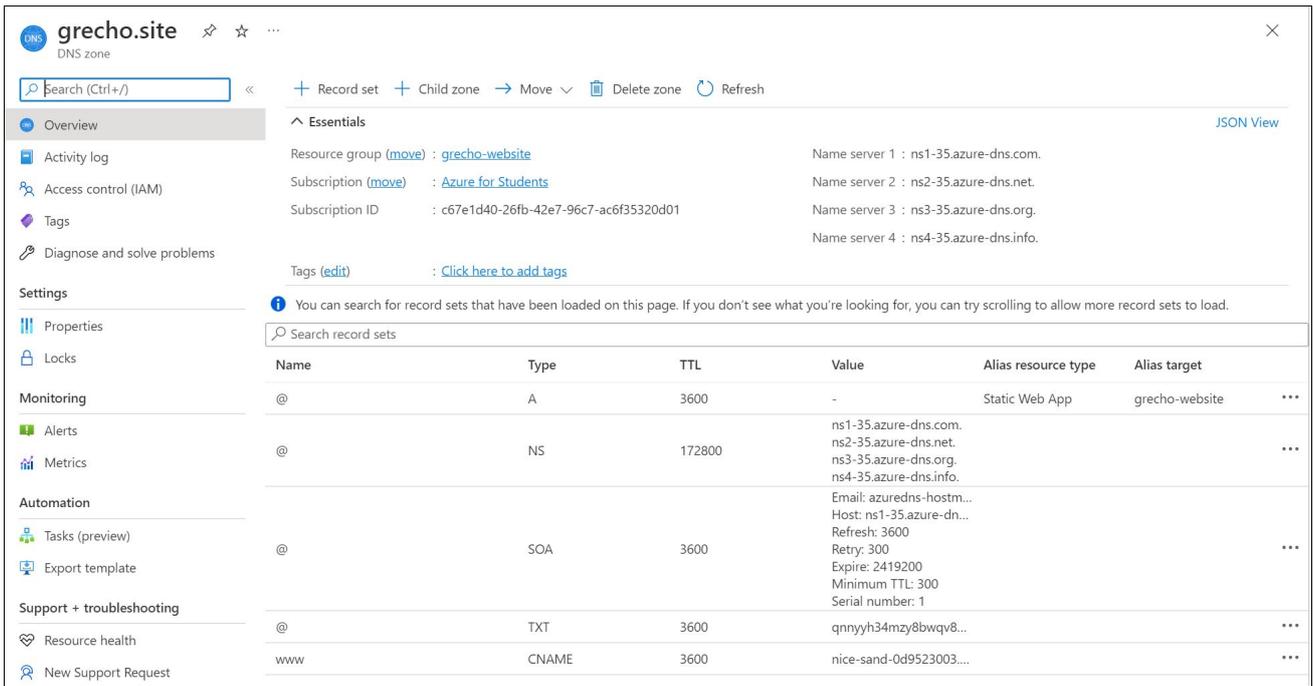


Figure 5: *Grecho* DNS zone setup on Azure

At a high-level, the following steps need to be performed, as an Azure user with global administrator permissions:

1. Create an Azure Key Vault instance.
2. Set up the Verifiable Credentials (also known as Verified ID) service
3. Register the *AdvertChain* application in Azure AD
4. Configure the Verifiable Credentials Service endpoint

The Azure Key Vault instance is needed to store the public and private keys to allow the VC service to sign and verify credentials.

Before we create any Cloud resource for the *AdvertChain* solution, we must first create a resource group. For this reason, go to the Resource Groups in the Azure portal menu and select Create.

Name: A unique name is required. In our case, we chose the name **advertchain-rg**.

Subscription: Choose the Azure subscription. In our case, we use the **Azure for Students** subscription.

Region: Choose any region that is suitable for you. In our case, we chose **West Europe**.

Finally, click Review + Create. It takes a few seconds to create the resource group.

Create a resource group ...

Basics Tags Review + create

Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. [Learn more](#) ↗

Project details

Subscription * ⓘ Azure for Students

Resource group * ⓘ advertchain-rg ✓

Resource details

Region * ⓘ (Europe) West Europe

Figure 6: Azure Resource Group for *AdvertChain*

To create an Azure Key Vault instance, go to the Key Vaults section of the Azure portal. On the Key Vault section, choose Create. On the Create key vault section provide the following information:

Name: A unique name is required. In our case, we chose the name **advertchain-kv**.

Subscription: Choose the Azure subscription.

Resource Group: Select **advertchain-rg**.

After providing the information above, select Create to create the Azure Key Vault instance.

After the Key Vault instance has been created, we need to set access policies for the Verifiable Credentials Admin user.

In the Azure portal, go to the **advertchain-kv** key vault instance. Under Settings, select Access policies. In Add access policies, under USER, select the account you use. For Key permissions, verify that the following permissions are selected:

1. Create
2. Delete
3. Sign

By default, Create and Delete are already enabled. Sign should be the only key permission you need to update.

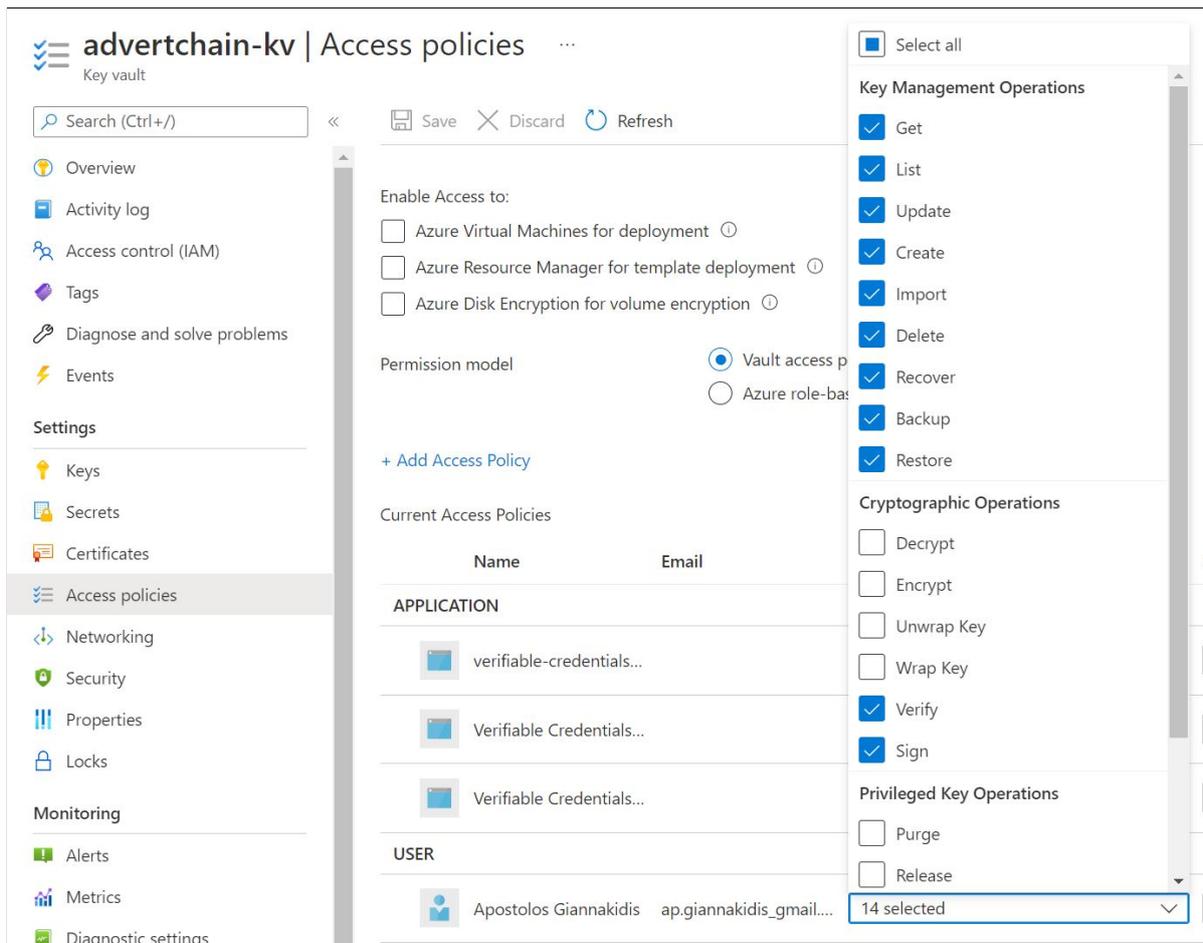


Figure 7: Azure Key Value User Access Policies

Finally, we must provide access to the Verifiable Credentials Service to the Key Value. To do this, select Access Policy and select the service principal Verifiable Credentials Service Request with AppId 3db474b9-6a0c-4840-96ac-1fceb342124. For Key permissions, select permissions Get and Sign. These permissions will allow Verifiable Credentials Service Request (Request Service API) to sign VC issuance and presentation requests. Finally, select Save to save the changes.

Note that the AppId 3db474b9-6a0c-4840-96ac-1fceb342124 is common for all tenants.

At this point we have everything we need to start setting up the Verifiable Credentials Service.

In the Azure portal, search for verified ID (previously named Verifiable Credentials). Then, select verified ID and from the left menu, select Getting started. Then, we must set up the organization details. The organization we will create will be the fake company organization: *Grecho*. So, to set up the organization we must provide the following information:

Organization name: **Grecho**

Domain: Enter **https://www.grecho.site**. This domain is added to a service endpoint in your DID document. The domain is what binds your DID to something tangible that

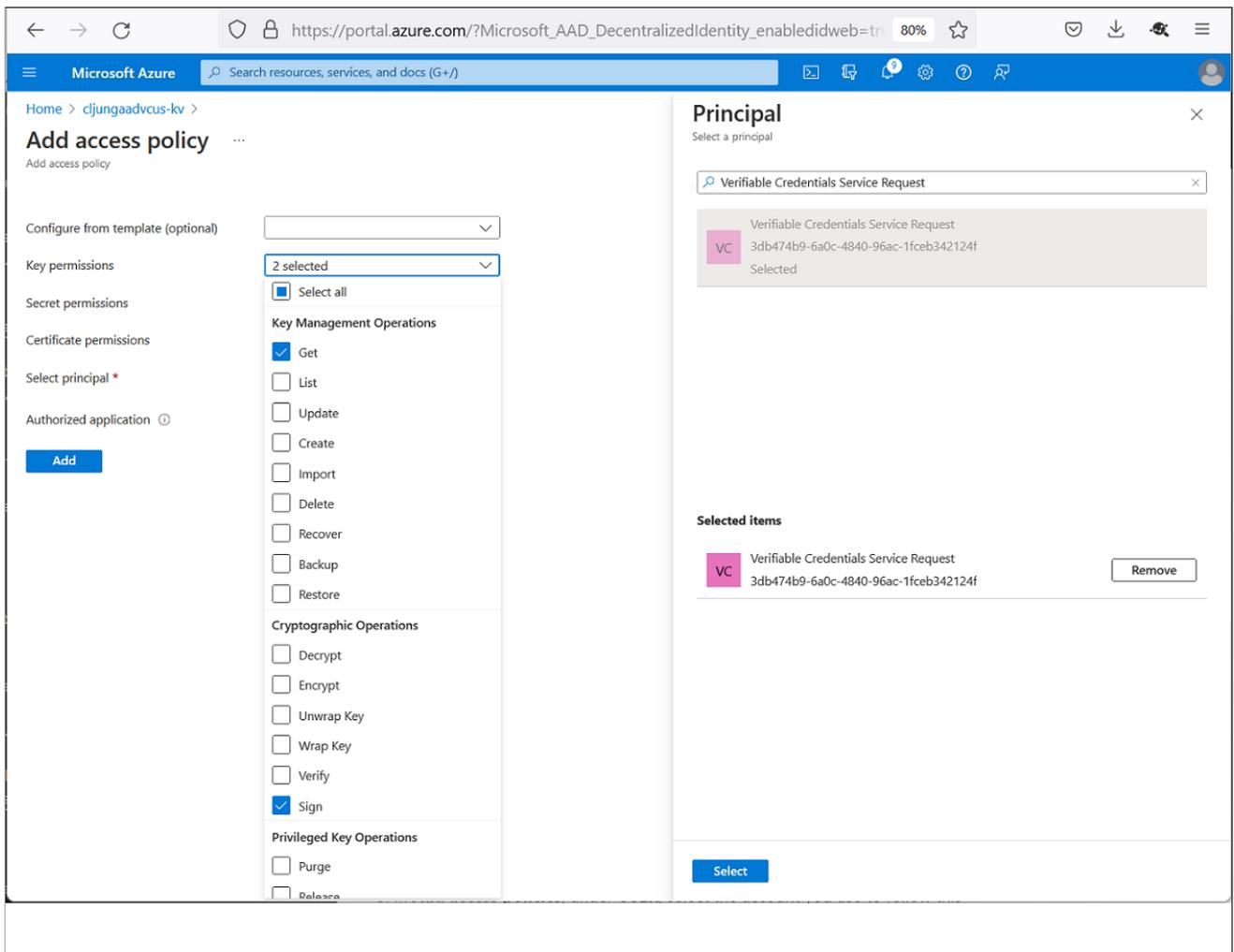


Figure 8: Access policy for the Verifiable Credentials Service Request service principal

the user might know about your business. Microsoft Authenticator and other digital wallets use this information to validate that your DID is linked to your domain. If the wallet can verify the DID, it displays a verified symbol. If the wallet can't verify the DID, it informs the user that the VC was issued by an unvalidated organization.

Key vault: Select the key vault **advertchain-kv** that you created earlier.

Under Advanced, choose the trust system that you want to use for your tenant. You can choose from either Web or ION. Web means your tenant uses did:web as the DID method and ION means it uses did:ion. In this Proof-of-Concept we will use the the **ION** method, which is a public, permissionless, decentralized DID overlay network that runs on top of the Bitcoin network. Finally, select Save.

Figure 9 shows the values required to setup the Azure VC Service for the PoC.

Next, we need to register the *AdvertChain* application in Azure AD so that it is authorized to get access tokens to issue and verify VCs. To get access tokens, register a web application and grant API permission for the API Verifiable Credential Request Service that you set up in the previous step.

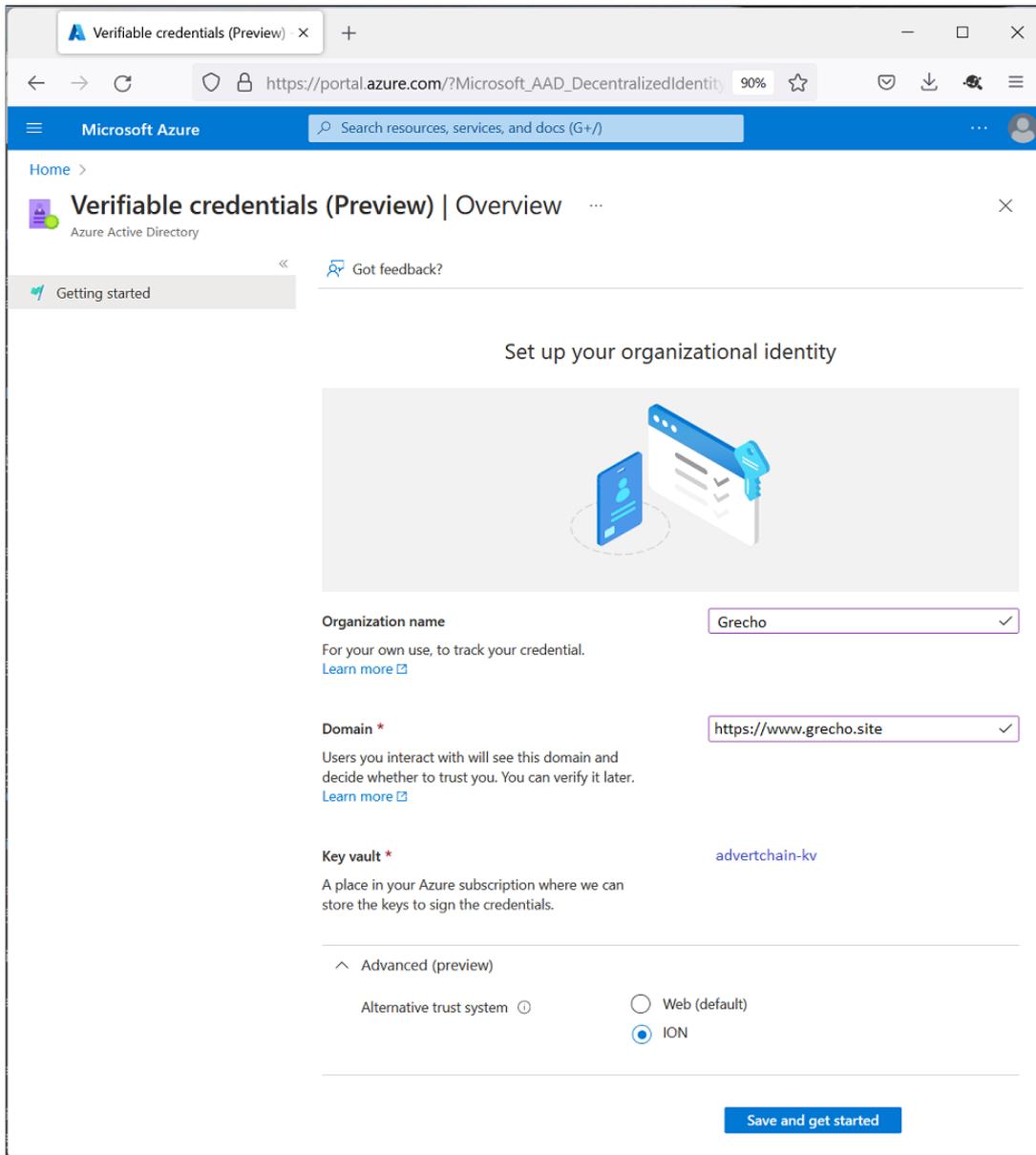


Figure 9: Setting up Verifiable Credentials service

In the Azure portal with your administrative account select your Azure Active Directory instance. Under Manage, select App registrations ->New registration. Enter a display name for your application. In our case we enter: **verifiable-credentials-app**. Select Register to create the application.

At this point we need to grant permissions to to app to get access tokens. In the verifiable-credentials-app application details page, select API permissions ->Add a permission. Select APIs my organization uses. Search for the service principal that you created earlier, Verifiable Credentials Service Request, and select it. Choose Application Permission, and expand *VerifiableCredential.Create.All*. Finally, select Add permissions and Grant admin consent for <your tenant name>.

Now we need to verify the organization's domain. In our case, this is www.grecho.site. In the Azure portal, navigate to the Verifiable credentials page and select Registration.

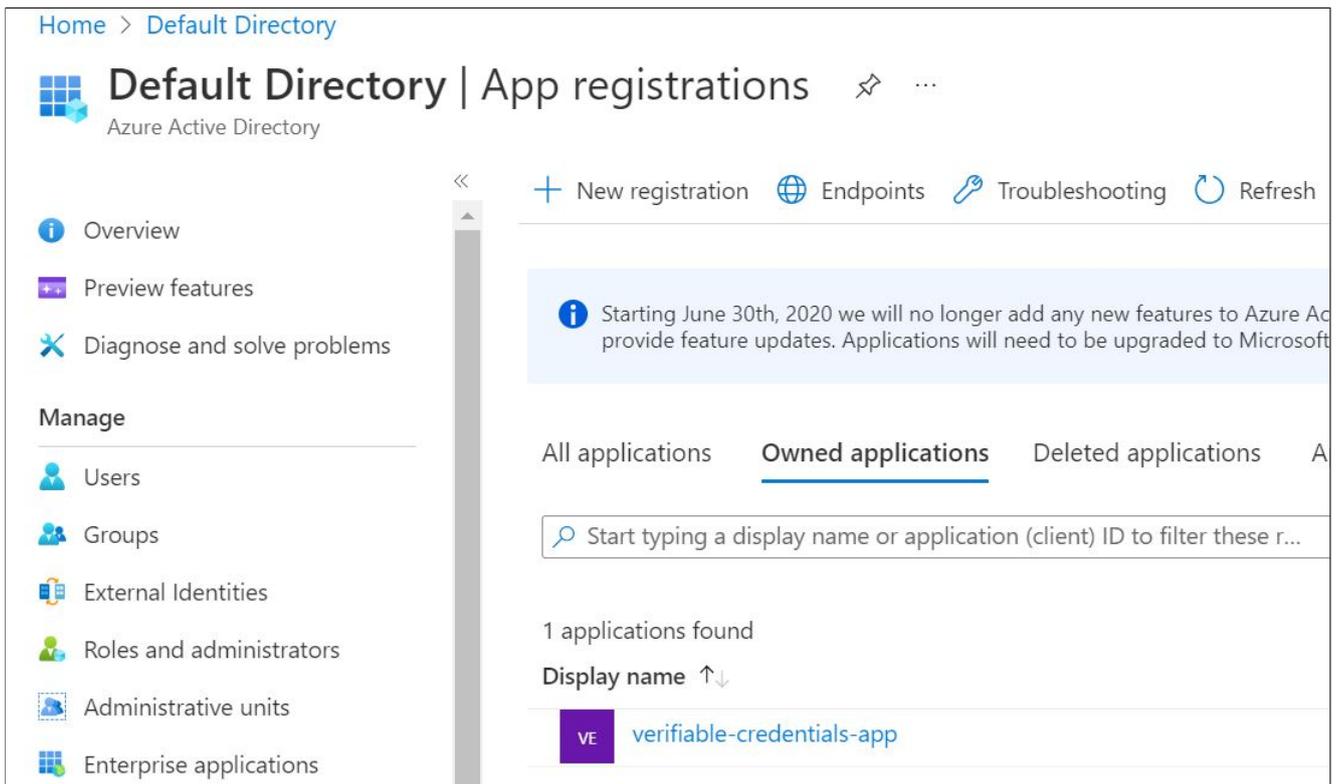


Figure 10: Registering a VC application in Azure AD

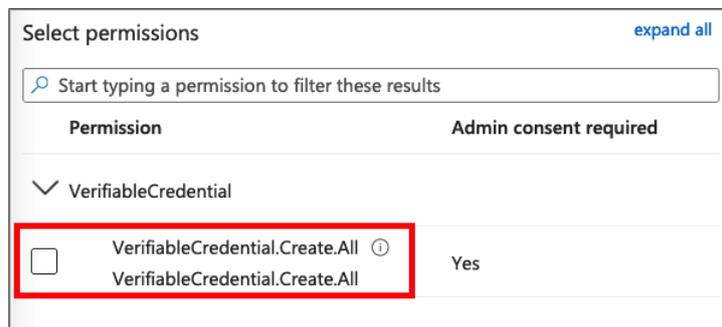


Figure 11: Granting permissions to get access tokens

Select Domain verification and download the JSON file. Then, deploy the JSON file on the hosting service of the [www.grecho.site](https://grecho.site) website. In our case, the URL for this file is:

<https://grecho.site/.well-known/did-configuration.json>

After making the `did-configuration.json` accessible on the domain, the domain publishing and verification will require up to **2** hours to complete.

The final thing that we need to configure is an Azure storage account in order to store the display rules JSON definitions for the Verifiable Credentials.

Note that this was needed during the public preview version of the service. After the service's GA release in early August 2022, the service allows for Managed Credentials. Managed Credentials no longer use of Azure Storage accounts to store the display rules

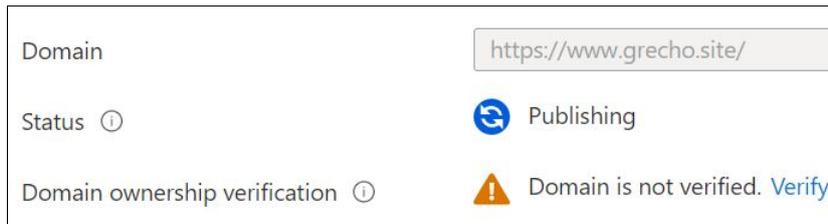


Figure 12: Waiting for domain verification to complete

JSON definitions. Thus, for new users it will be much simpler to store their display rules JSON definition files.

At this point the VC service is configured. The next step is to create the Verifiable Credential issuance service. Select Credentials from the left blade, as shown in the figure 13. Then click on the "Create Credential" button. When asked about the credential type, select "Custom Credential", as shown in the figure 14

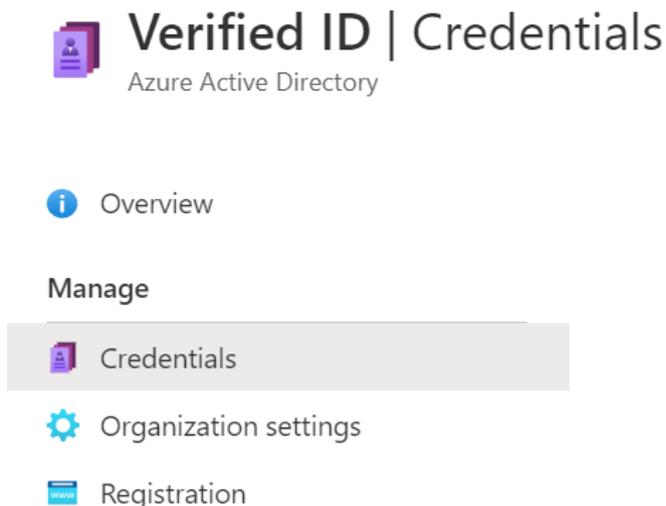


Figure 13: Verified ID Credentials menu

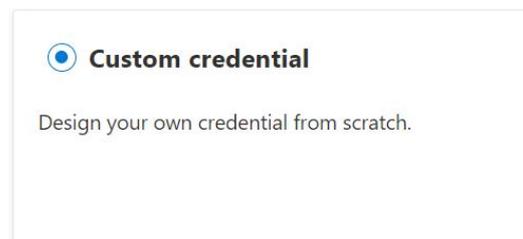


Figure 14: Custom Credential Type

Next, the display rules JSON definition files must be configured. Verifiable credentials definitions are made up of two components, **display definitions** and **rules definitions**. A display definition controls the branding of the credential and styling of the claims. A rules definition determines what users need to provide before they receive a verifiable credential. For more information on how to create your own custom **display definition** and **rules definition** file go to: <https://docs.microsoft.com/en-us/azure/active-directory/verifiable-credentials/rules-and-display-definitions-model>.

The *AdvertChain* Proof-of-Concept, uses the files that are accessible via:

<https://github.com/maestros/advertchain-portal/tree/main/vc-contract>.

At the point, the Verifiable Credential is ready to be used. In order to be able to use the VC we need to first gather some information about the environment and the VC that you created. To gather the information we need, follow these steps:

1. In Verifiable Credentials, select Issue credential.

2. Copy the authority, which is the Decentralized Identifier, and record it for later.
3. Copy the manifest URL. It's the URL that Authenticator evaluates before it displays to the user VC issuance requirements. Record it for later use.
4. Copy your Tenant ID, and record it for later. The Tenant ID is the guid in the manifest URL highlighted in red.

The screenshot in figure 15 shows the information that needs to be copied from the newly created verified credential.

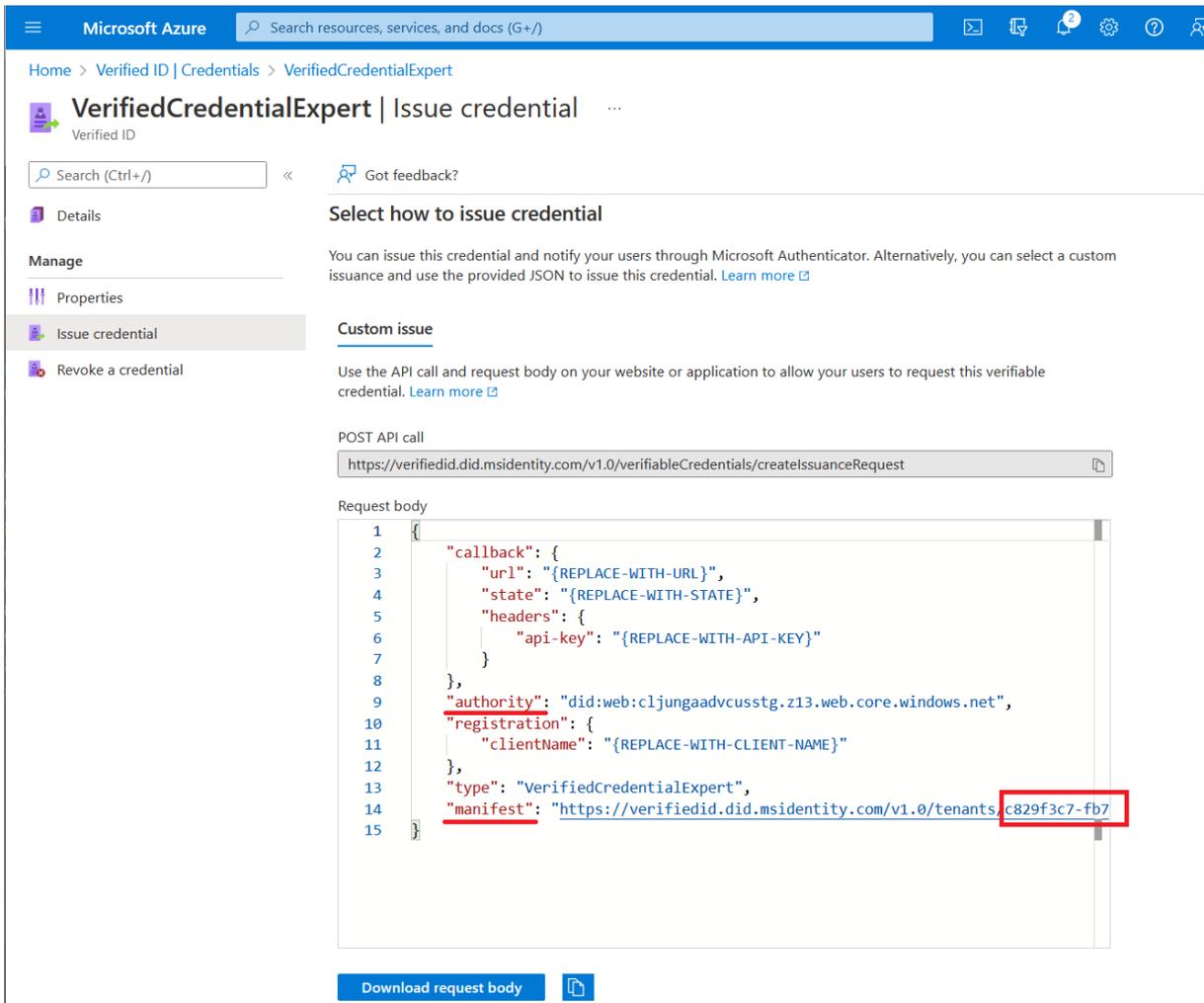


Figure 15: Selecting information from the new VC

4.1.3 Setting up the Verifiable Credentials Issuer

Now that we have created and verified the domain and the website for *Grecho* as well as the Azure AD Verifiable Credentials service, we can deploy the *Grecho* VC issuer app.

Note that the VC issuer page is not part of the *AdvertChain* solution. It is however, required for the end-to-end demonstration of the user journey.

To this end, the following source code repo has been created:

<https://github.com/maestros/grecho-vc-issuer>.

Clone this repo by entering the following command:

```
git clone git@github.com:maestros/grecho-vc-issuer.git
```

Then, edit the **1-node-api-idthokenhint/config.json** file and add the Azure Tenant and VC information that was gathered before.

Finally, deploy this app on a hosting service. In our case, this app was deployed on Azure App Service. The deployment was done via the Azure extension of the Microsoft Visual Studio Code application. This deployment process is shown in the figure 16.

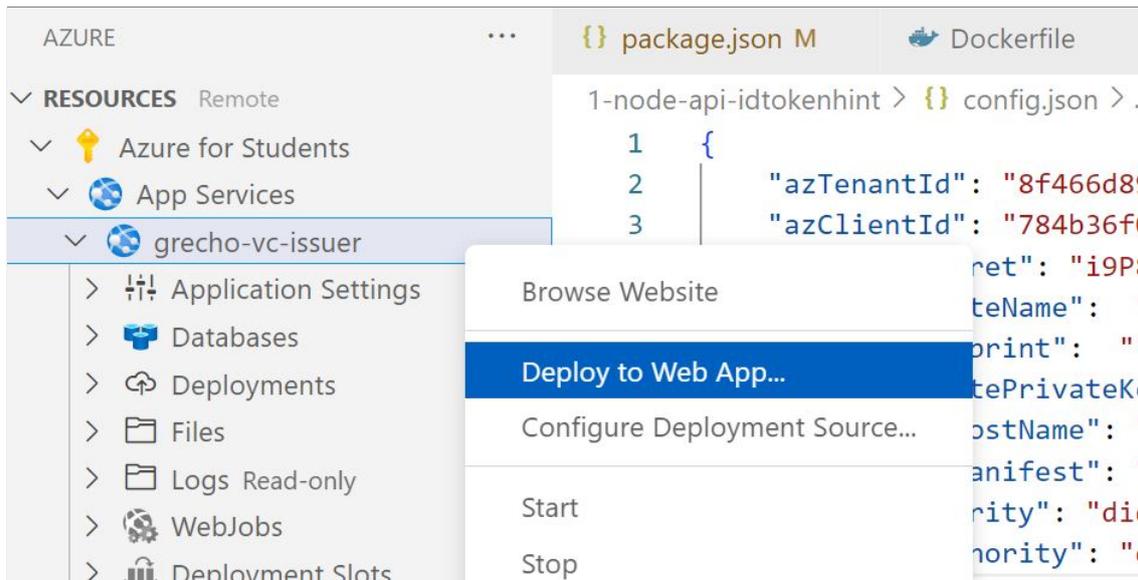


Figure 16: Deploying the VC issuer app on Azure App Service using Visual Studio Code

In this Proof-of-Concept, the *Grecho* VC issuer app is accessible at: <https://grecho-vc-issuer.azurewebsites.net/>.

4.2 Firebase Cloud

Google's Firebase was chosen as the PaaS platform to deploy the *AdvertChain* NFT minter web application and its backend serverless services.

Go to the Firebase console on <https://console.firebase.google.com/> and select **Add Project**. We must give a unique name for the project. In our case we chose **advertchain-firebase**. Then, make sure to upgrade to the **Blaze** Pay as you go pricing plan. For this Proof-of-Concept there is not going to be any cost, however, just to be safe set up a budget for €2 per month. From the project's settings note the Web API key in a notepad.

Then, create a new Firebase app. In our case, we chose **firebase-demo**. After the app's creation, copy the App ID in a notepad. Then create a Firebase Hosting site and give it the same name. This will be used in the application's domain generated by Firebase. Then link the app with the Firebase hosting site as shown in figure 18.

Then, select **Realtime Database** and define the rules as show in figure 19.

Note that we chose to disallow read and write operation after the 31st of December 2023. You may choose to define different rules.

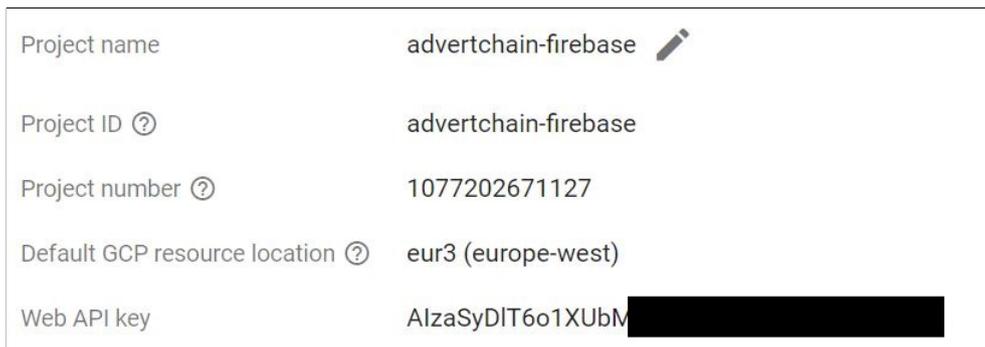


Figure 17: Firebase Project Settings

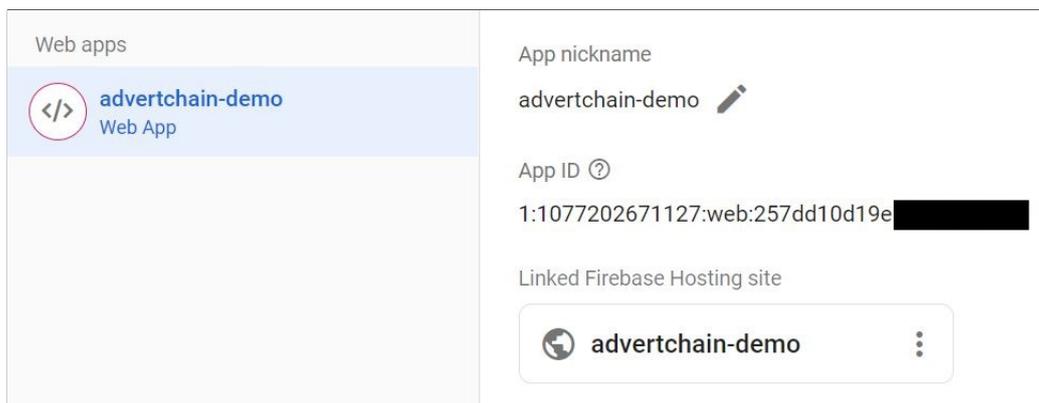


Figure 18: Firebase App Settings



Figure 19: Firebase Realtime Database Rules

Finally, go to project's setting and scroll at the bottom to find the **SDK setup and configuration**. Make sure that npm is selected and then copy the Javascript code that is provided for the initialization of our Firebase project. In our case, the code is shown in figure 20.

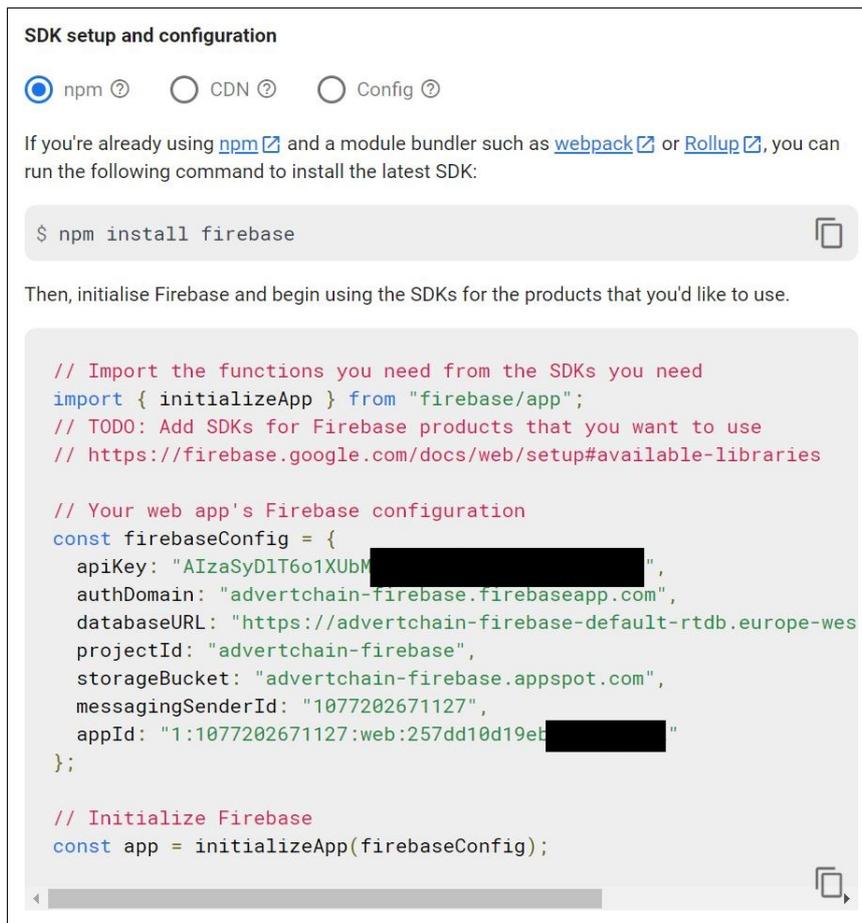


Figure 20: Firebase App Initialization Code

4.3 Pinata Cloud

This project requires storing the NFT token metadata on IPFS. The problem with IPFS nodes is that infrequently accessed content is removed from their cache when their garbage collection process runs. To make sure that the NFT token metadata remains available on the IPFS network we need to pin the content. Using Pinata to pin the *AdvertChain* NFT metadata we can ensure that the content will always be online, without having to worry about maintaining and monitoring our own IPFS nodes. To programmatically interact with Pinata, we must create a new API key. After logging in to Pinata, go to <https://app.pinata.cloud/keys> and select + **New Key**. Make sure you select all permission for Pinning and Data, however we do not need Admin permissions for this key. Choose the name **AdvertChain API** and then select Create.

In the next screen, the API secret key will be shown. Copy the API secret in a notepad because we will need it later when we configure the *AdvertChain* NFT minter application.

4.4 Alchemy Web3

Finally, we need an API key for the Alchemy Web3 provider. Log in to your Alchemy account and visit <https://dashboard.alchemyapi.io/> and create a new app. We must provide a unique name. In our case we chose **Advertchain**. Then choose the **Polygon Mumbai** network and click Create App. Then select the **API KEY** link to see the API keys, as shown in figure 24. Copy the HTTPS key value to a notepad.

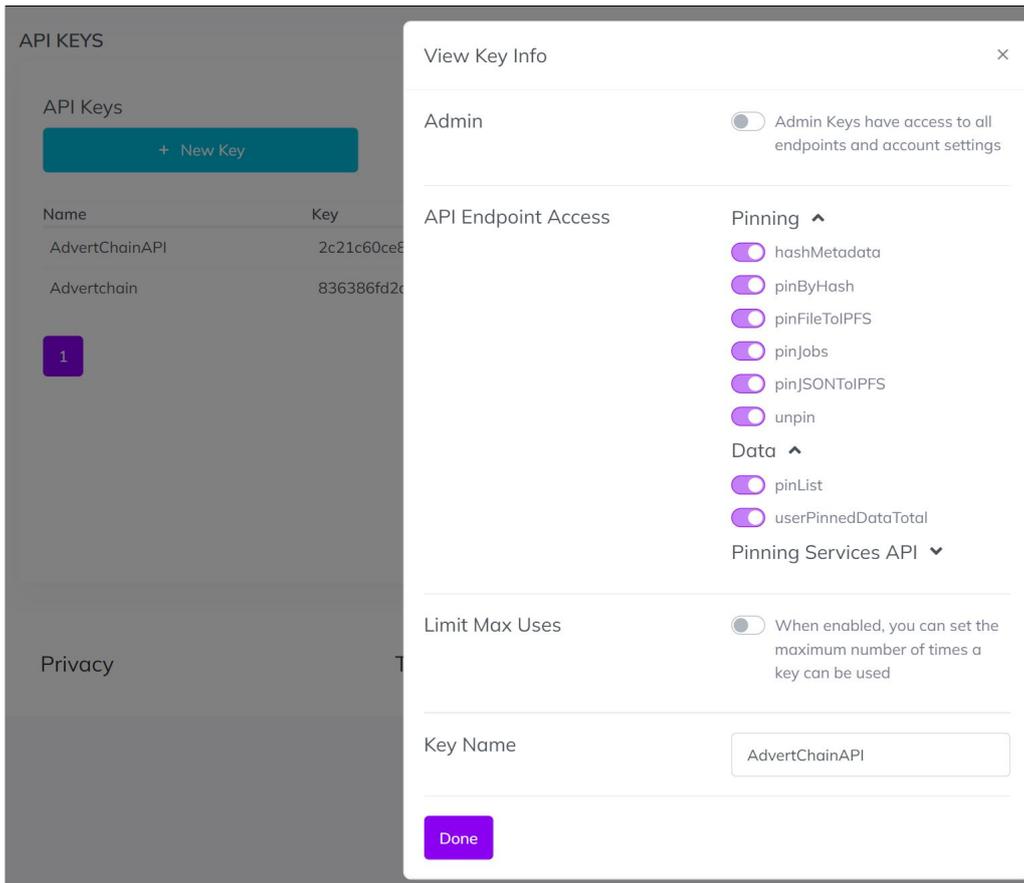


Figure 21: Creating a new Pinata API key

4.5 NoCode API for screenshot generation

In order to create NFTs for the online job posts, we must be able to programmatically retrieve a screenshot for the job posts. It is traditional, but not mandatory, each NFT to have a linked image. To get the screenshot we will use the NoCodeAPI platform that provides an easy way to create APIs for various use cases. One of their supported APIs is the screenshot capturing. Go to <https://app.nocodeapi.com/dashboard> and activate the **Website Screenshot** app. Copy the API endpoint to a notepad.



Figure 22: Copying the Pinata API key secret



Figure 23: Getting the Alchemy Web3 API Key



Figure 24: Website Screenshot API provided by NoCode API

5 LinkedIn Identity Federation Setup

AdvertChain leverages Identity Federation to allow the user to log in via the LinkedIn social login capability. To achieve Identity Federation via LinkedIn, a few steps need to be configured and implemented.

First, create a new LinkedIn developer app on <https://developer.linkedin.com/>. In our case, we chose the name **AdvertChain**, as shown in figure 25. After the app is created copy the client id to a notepad.

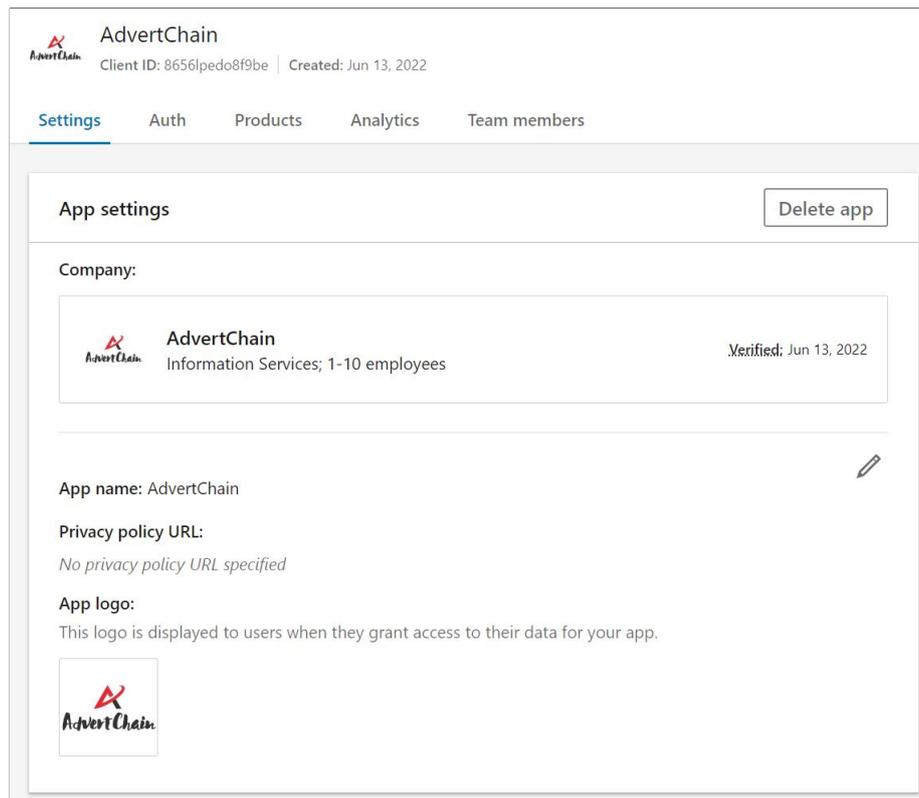


Figure 25: The AdvertChain LinkedIn developer app

Then, go to the Auth menu and copy the Client Secret value to a notepad. Now, the important step to enable Identity Federation is add an authorized redirect URL for the *AdvertChain* app. Go to the OAuth 2.0 settings and edit the list of Authorized redirect URLs. Add the URL of the callback function that will handle the Identity Federation in the app. In our case, the URL is: **<https://advertchain-demo.web.app/callback>**. This URL maps to a Firebase Cloud Function that we will deploy in section 7. Section 7 will provide more information about how the callback function works. Figure 25 shows the OAuth callback URL for the *AdvertChain* app.

Finally, make sure that both **r_emailaddress** and **r_liteprofile** are in scope. These will authorize *AdvertChain* to retrieve the federated user's email and full name.

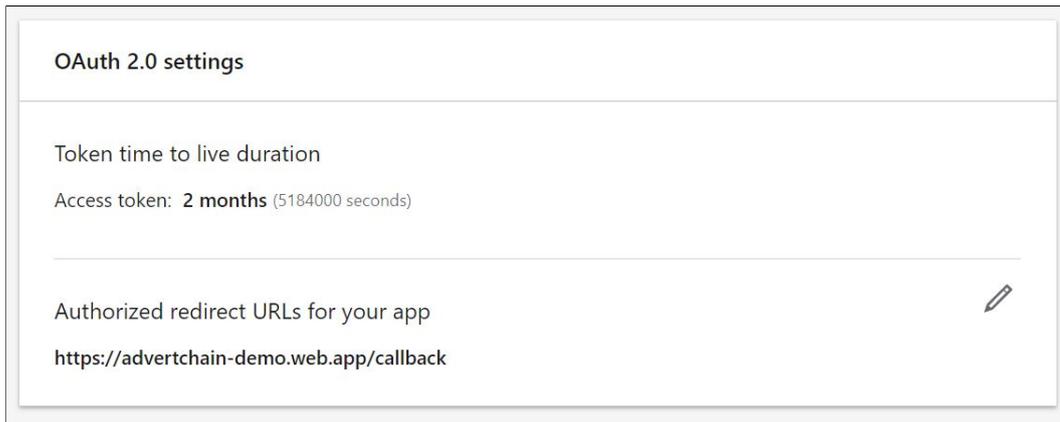


Figure 26: The AdvertChain LinkedIn app OAuth 2.0 settings

6 NFT Deployment on Polygon

The *AdvertChain* ERC-721 NFT Smart Contract has been deployed on the Mumbai Polygon/Matic testnet. The compilation, deployment and verification of the Smart Contract has been automated via HardHat.

In order for the Smart Contract to be recompiled, redeployed and reverified a set of steps must be run. Before running the steps, the following software tools and packages must be installed on the local system:

1. Git 2.34.0 (or higher)
2. NodeJs 16.15.0 (or higher)
3. NPM 8.13.2 (or higher)

If you have not cloned the *AdvertChain* repo yet, open a terminal window and enter:

```
git clone git@github.com:maestros/advertchain-portal.git
```

Next, make sure you use the latest npm:

```
npm install npm@latest -g
```

Next, install all dependencies:

```
npm install --save
```

In the **contract** folder, create a **.env** file as shown in the figure 27. Three environmental variables are needed in this file:

1. **PRIVATE_KEY**, which is your MetaMask private key
2. **POLYGONSCAN_API_KEY**, which is the API key from Polygonscan

Hardhat is configured via the **hardhat.config.js** file. Hardhat has been configured to allow the compilation, deployment and verification for the *AdvertChain* Smart Contract. The Hardhat configuration is shown in the Figure 28.

```

contract > .env
1 // Metamask Private Key
2 PRIVATE_KEY = "4700f15f34061ebae75d1aa6b205d5da96bf561fd696824bd05424661a746f23"
3
4 //Etherscan API key
5 ETHERSCAN_KEY = "C844T5SFFGQI87157Q7PESS18U8CTATARP"

```

Figure 27: NFT deployment & verification environmental variables

```

9  const { PRIVATE_KEY, ETHERSCAN_KEY } = process.env;
10 module.exports = {
11   solidity: "0.8.12",
12   defaultNetwork: "matic",
13   networks: {
14     hardhat: {},
15     matic: {
16       url: "https://rpc-mumbai.maticvigil.com",
17       accounts: [`0x${PRIVATE_KEY}`]
18     }
19   },
20   paths: {
21     sources: "./nft",
22     tests: "./test",
23     cache: "./cache",
24     artifacts: "./artifacts"
25   },
26   etherscan: {
27     apiKey: ETHERSCAN_KEY
28   }
29 }

```

Figure 28: Hardhat configuration

To compile the *AdvertChain* Smart Contract via Hardhat, enter:

```
npx hardhat compile
```

or if there are no code changes in the Smart Contract and you need to recompile:

```
npx hardhat compile --force
```

Every time you compile a new version of the *AdvertChain* NFT Smart Contract, you must copy the compiled artifact to the NFT minter app. This is because the *AdvertChain* NFT minter app depends on the Smart Contract. To achieve this enter a command similar to the following, after making sure that the file paths are correct for your system:

```
copy c:\repos\advertchain-portal\contract\artifacts\nft\
AdvertChain.sol\AdvertChain.json c:\repos\advertchain-portal\
nft-minter\client\src\AdvertChain.json
```

Next, in order to deploy the compiled *AdvertChain* NFT Smart Contract to the Mumbai/Matic testnet, enter the following command:

```
npx hardhat run scripts/deploy.js --network matic
```

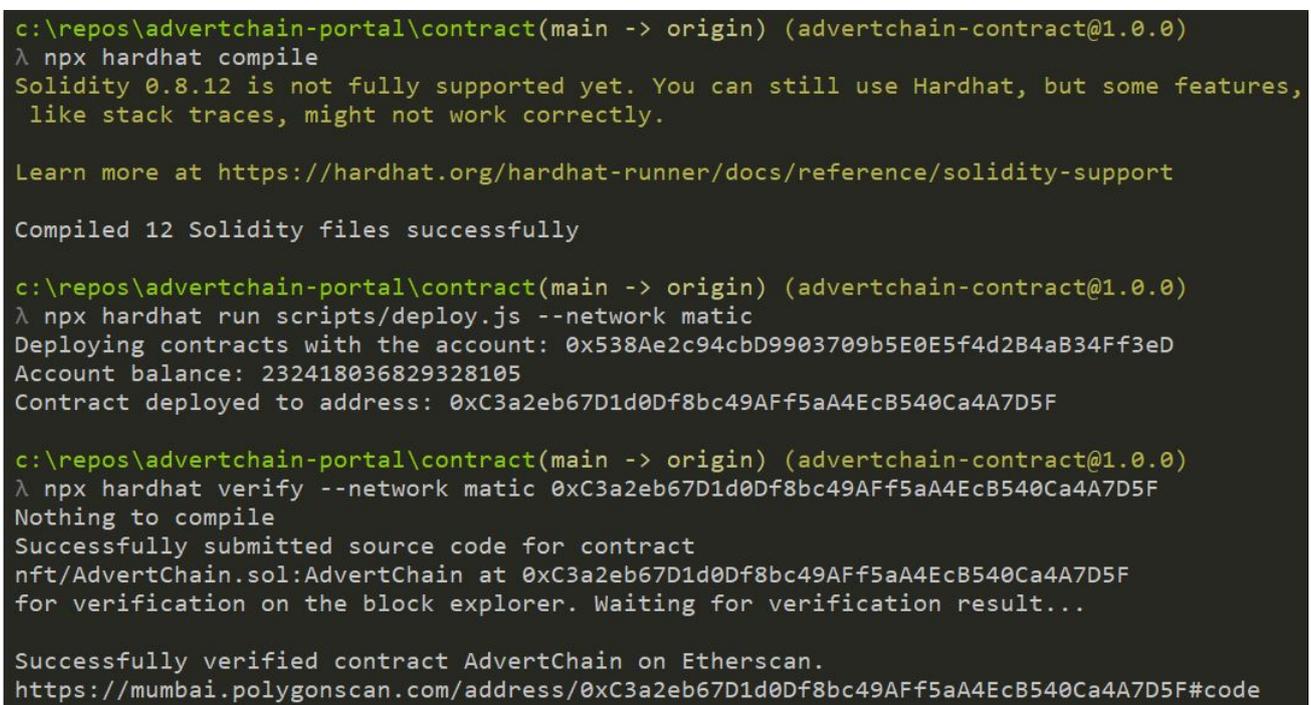
The output of the deployment command will display the Mumbai testnet address where our contract was deployed at. Copy this address on a notepad.

The last step is to verify the *AdvertChain* NFT Smart Contract source code. Before we proceed to the verification, it is important to first wait for 1 minute after we deploy the contract. This is required because the Blockchain needs some time to process the new block. After a minute or two, enter the following command, after replacing the correct Matic contract address:

```
npx hardhat verify --network matic <matic-contract-address>
```

The output of the verify command will display the link to Polygonscan with the verified contract source code.

The whole process of compiling the contract, deploying it to Polygon/Matic and verifying it via HardHat is shown in figure 30.



```
c:\repos\advertchain-portal\contract(main -> origin) (advertchain-contract@1.0.0)
λ npx hardhat compile
Solidity 0.8.12 is not fully supported yet. You can still use Hardhat, but some features,
like stack traces, might not work correctly.

Learn more at https://hardhat.org/hardhat-runner/docs/reference/solidity-support

Compiled 12 Solidity files successfully

c:\repos\advertchain-portal\contract(main -> origin) (advertchain-contract@1.0.0)
λ npx hardhat run scripts/deploy.js --network matic
Deploying contracts with the account: 0x538Ae2c94cbD9903709b5E0E5f4d2B4aB34Ff3eD
Account balance: 232418036829328105
Contract deployed to address: 0xC3a2eb67D1d0Df8bc49Aff5aA4EcB540Ca4A7D5F

c:\repos\advertchain-portal\contract(main -> origin) (advertchain-contract@1.0.0)
λ npx hardhat verify --network matic 0xC3a2eb67D1d0Df8bc49Aff5aA4EcB540Ca4A7D5F
Nothing to compile
Successfully submitted source code for contract
nft/AdvertChain.sol:AdvertChain at 0xC3a2eb67D1d0Df8bc49Aff5aA4EcB540Ca4A7D5F
for verification on the block explorer. Waiting for verification result...

Successfully verified contract AdvertChain on Etherscan.
https://mumbai.polygonscan.com/address/0xC3a2eb67D1d0Df8bc49Aff5aA4EcB540Ca4A7D5F#code
```

Figure 29: *AdvertChain* NFT contract compilation, deployment and verification

As it can be seen from the screenshot in Figure 30, the final version of the *AdvertChain* NFT Smart Contract has been deployed (and verified) on the Mumbai testnet address:

0xC3a2eb67D1d0Df8bc49Aff5aA4EcB540Ca4A7D5F

Copy this address to a notepad, as it will be needed later to configure *AdvertChain*.

Since we verified the Smart Contract, its source code can be accessible via Polygonscan: <https://mumbai.polygonscan.com/address/0xC3a2eb67D1d0Df8bc49Aff5aA4EcB540Ca4A7D5F#code>. Using this Polygonscan link we can also interact manually with the deployed *AdvertChain* ERC-721 Smart Contract.

In this PoC, the Smart Contract name is **AdvertChain** and its symbol is **ADCN**. In case you wish to change the name of the contract or its symbol, go to the **AdvertChain.sol**

file in the repo and change the arguments of the ERC-721 constructor, as shown in Figure 30.

```
18 constructor() ERC721("AdvertChain", "ADCN") {
19     _minter = msg.sender;
20 }
```

Figure 30: *AdvertChain* NFT contract constructor

The screenshot on Figure 31 shows the final version of the deployed *AdvertChain* contract on Polygonscan at the address: **0xC3a2eb67D1d0Df8bc49AFf5aA4EcB540Ca4A7D5F**.

The screenshot displays the Polygonscan interface for the contract **0xC3a2eb67D1d0Df8bc49AFf5aA4EcB540Ca4A7D5F**. The interface includes a search bar, navigation tabs (Home, Blockchain, Tokens, Misc, Testnet), and a sidebar with network options (Polygon Mainnet, Mumbai Testnet). The main content area shows the contract overview, including the balance (0 MATIC) and the contract creator's address. Below this, there are tabs for Transactions, ERC-20 Token Txns, Contract (selected), and Events. The Contract tab is active, showing the source code verification status as "Contract Source Code Verified (Exact Match)". The contract details include the name "AdvertChain", compiler version "v0.8.12+commit.f00d7308", and optimization settings "No with 200 runs". The source code is displayed in a code editor, showing the constructor function and various imports and comments.

```
1 // SPDX-License-Identifier: MIT
2 // OpenZeppelin Contracts (last updated v4.7.0) (token/ERC721/ERC721.sol)
3 // Author: Apostolos Giannakidis (ap.giannakidis@gmail.com)
4 // Version: 1.6
5
6 pragma solidity ^0.8.12;
7
8 import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
9 import "@openzeppelin/contracts/utils/Counters.sol";
10
11 contract AdvertChain is ERC721URIStorage {
12     // The contract owner and the only address that is allowed to mint new NFTs.
13     address private _minter;
14     using Counters for Counters.Counter;
15     Counters.Counter private _tokenIds;
16     mapping(string => address) private _jobIds;
17
18     constructor() ERC721("AdvertChain", "ADCN") {
19         _minter = msg.sender;
20     }
21
22     /*
23     * address recipient: the recipient that will receive this newly minted NFT
24     * string tokenURI: the URI that points to the JSON document that contains the NFT metadata
25     * string jobId: the unique id of the job posting (typically: the <job-ID> of the job post)
```

Figure 31: Verified *AdvertChain* NFT contract on Polygonscan

7 AdvertChain Software

The following software packages are required to build *AdvertChain* locally and to deploy it to the Cloud:

1. Git 2.34.0 (or higher)
2. NodeJs 16.15.0 (or higher)
3. NPM 8.13.2 (or higher)
4. Firebase CLI

Before we proceed with the configuration, installation and deployment steps for the *AdvertChain* software, we must first install and configure the Firebase CLI.

To install the Firebase CLI, go to <https://firebase.google.com/docs/cli>, download the binary for your Operating System and install it.

After installing the Firebase CLI, you must authenticate. Enter the following command to login:

```
firebase login
```

When asked for you account credentials, enter the ones that you used to create the Firebase account in chapter 3.

Then, make sure that the file **.firebaserc** has been created. If not, then create the file manually and add this content:

```
{
  "projects": {
    "default": "advertchain-firebase"
  },
  "targets": {}
}
```

In your case, change the default project name according to your project's name. Also, change the Firebase hosting site accordingly in the **firebase.json** file. In our case, this value is set to "**advertchain-demo**".

Next, test that the CLI is properly installed and accessing your account by listing your Firebase projects. Run the following command:

```
firebase projects:list
```

The displayed list should be the same as the Firebase projects listed in the [Firebase console](#).

Finally, run the following command:

```
firebase use --add
```

When prompted, select your Firebase project. In our case, that is **advertchain-firebase**.

Now, we can proceed to the configuration, installation and deployment steps for the *AdvertChain* software.

The *AdvertChain* software consists of two parts:

1. the NFT minter web app (client front-end)
2. the backend serverless functions

7.1 Deploying the AdvertChain backend

To deploy *AdvertChain* we will first deploy the backend.

Go to the folder **advertchain-portal\nft-minter\functions**. The *AdvertChain* backend requires the following libraries and third-party dependencies:

1. @alch/alchemy-web3@1.4.6
2. @azure/msal-node@1.12.0
3. @decentralized-identity/ion-tools@0.1.1
4. @pinata/sdk@1.1.26
5. axios@0.27.2
6. dotenv@16.0.1
7. body-parser@1.20.0
8. cheerio@1.0.0-rc.12
9. cookie-parser@1.4.6
10. cors@2.8.5
11. crypto@1.0.1
12. ejs@3.1.8
13. express-session@1.17.3
14. express@4.18.1
15. firebase-admin@11.0.0
16. firebase-functions@3.22.0
17. lodash@4.17.21
18. morgan@1.10.0
19. node-fetch@3.2.9
20. npm@8.13.2
21. path@0.12.7
22. superagent@3.8.3
23. uglify-js@3.16.2 extraneous
24. web3@1.7.4

To install these dependencies, go to the **functions** folder and run the following command:

```
npm install -- save
```

Before deploying the backend to Google’s Firebase Cloud, we must first create a `.env` file with the environmental variables that the backend requires. Specifically, the following environmental variables are needed:

1. **LINKEDIN_REDIRECT_URI**, which is the public URL of the serverless function that handles the LinkedIn Identity Federation redirect callback. This is the same URL that we configured in the *AdvertChain* developer app on LinkedIn in chapter 5.
2. **LINKEDIN_CLIENT_ID**, this is the client id of the *AdvertChain* developer app on LinkedIn.
3. **LINKEDIN_CLIENT_SECRET**, this is the client secret of the *AdvertChain* developer app on LinkedIn.
4. **PINATA_KEY**, which is the API key for Pinata.
5. **PINATA_SECRET**, which is the API secret for Pinata.
6. **ADVERTCHAIN_CONTRACT**, which is the Mumbai testnet address of the deployed *AdvertChain* ERC-721 contract, as deployed in chapter 6.
7. **WEB3_PRIVATE_KEY**, which is the private key of the MetaMask wallet. Extract this key from MetaMask.
8. **ALCHEMY_KEY**, which is the Alchemy HTTPS URL that contains the API key for the Mumbai testnet.
9. **AZURE_DID_API_KEY**, which is the API key for the Azure DID/VC service.
10. **ADVERTCHAIN_FIREBASE_FUNCTION_BASE_URI**, which is the base URI of this Firebase project’s serverless Cloud functions. In our case, this is set to <https://us-central1-advertchain-firebase.cloudfunctions.net/api/>. This is displayed by Firebase after the backend gets deployed.
11. **ADVERTCHAIN_FIREBASE_API_KEY**, which is the API key for Firebase, as shown in the Firebase App Initialization Code in subsection 4.2.
12. **ADVERTCHAIN_FIREBASE_AUTH_DOMAIN**, which is the Firebase auth domain, as shown in the Firebase App Initialization Code in subsection 4.2.
13. **ADVERTCHAIN_FIREBASE_PROJECT_ID**, which is the Firebase project id, as shown in the Firebase App Initialization Code in subsection 4.2.
14. **ADVERTCHAIN_FIREBASE_APP_ID**, which is the Firebase app id, as shown in the Firebase App Initialization Code in subsection 4.2.
15. **NOCODEAPI_SCREENSHOP_API**, which is the API endpoint for screenshot capturing via the NoCode API platform.

Figure 32 shows the environmental variables defined in the PoC `.env` file.

To test the backend locally, before deploying to the Firebase Cloud, run the following command from the **functions** folder:

```
firebase serve
```

```

nft-minter > functions > .env
1 LINKEDIN_REDIRECT_URI = "https://advertchain-demo.web.app/callback"
2 LINKEDIN_CLIENT_ID = "86561pedo8f9be"
3 LINKEDIN_CLIENT_SECRET = "██████████"
4
5 PINATA_KEY = "2c21c60ce8██████████"
6 PINATA_SECRET = "d1c00e082c2e3d81ed86d7376b4804██████████"
7
8 ADVERTCHAIN_CONTRACT = "0xA73a74466a56c2EDe5D0F9f25AE597085dA23d4f"
9 WEB3_PRIVATE_KEY = "0x4700f15f34061ebae75d1aa6b205d5da96bf██████████"
10 ALCHEMY_KEY = "https://eth-goerli.g.alchemy.com/v2/qXXMWG0██████████"
11
12 AZURE_DID_API_KEY = "8294beaa-a7f9-490c-██████████"
13
14 ADVERTCHAIN_FIREBASE_FUNCTION_BASE_URI = "https://us-central1-advertchain-firebase.cloudfunctions.net/api/"
15 ADVERTCHAIN_FIREBASE_API_KEY = "AIzaSyDlT6o1XUbM4██████████"
16 ADVERTCHAIN_FIREBASE_AUTH_DOMAIN = "advertchain-firebase.firebaseio.com"
17 ADVERTCHAIN_FIREBASE_PROJECT_ID = "advertchain-firebase"
18 ADVERTCHAIN_FIREBASE_APP_ID = "1:1077202671127:web:257dd10d██████████"
19
20 NOCODEAPI_SCREENSHOP_API = "https://v1.nocodeapi.com/agiannakidis/screen/T██████████/screenshot"

```

Figure 32: *AdvertChain* backend environmental variables

If there are no compilation or runtime issues, *AdvertChain* will be deployed on localhost. Then, proceed to deploy the *AdvertChain* Cloud functions on Firebase by entering the following command:

```
firebase deploy
```

Figure 33 shows the Firebase deployment commands in the PoC's environment.

```

C:\repos\advertchain-portal\nft-minter\functions(main -> origin)
λ firebase serve

=== Serving from 'c:\repos\advertchain-portal\nft-minter'...

+ functions: Using node@16 from host.
i functions: Watching "c:\repos\advertchain-portal\nft-minter\functions" for Cloud Functions...
i hosting[advertchain-demo]: Serving hosting files from: client/build
+ hosting[advertchain-demo]: Local server: http://localhost:5000
+ functions[us-central1-api]: http function initialized (http://localhost:5001/advertchain-firebase/us-central1/api)
Shutting down...
i functions: Stopping Functions Emulator

c:\repos\advertchain-portal\nft-minter\functions(main -> origin)
λ firebase deploy

=== Deploying to 'advertchain-firebase'...

i deploying functions, hosting
i functions: ensuring required API cloudfunctions.googleapis.com is enabled...
i functions: ensuring required API cloudbuild.googleapis.com is enabled...
i artifactregistry: ensuring required API artifactregistry.googleapis.com is enabled...
+ artifactregistry: required API artifactregistry.googleapis.com is enabled
+ functions: required API cloudfunctions.googleapis.com is enabled
+ functions: required API cloudbuild.googleapis.com is enabled
i functions: preparing codebase default for deployment
i functions: Loaded environment variables from .env
i functions: preparing functions directory for uploading...
i functions: packaged c:\repos\advertchain-portal\nft-minter\functions (378.76 KB) for uploading
+ functions: functions folder uploaded successfully
i hosting[advertchain-demo]: beginning deploy...
i hosting[advertchain-demo]: found 34 files in client/build
+ hosting[advertchain-demo]: file upload complete
i functions: updating Node.js 16 function api(us-central1)...
+ functions[api(us-central1)] Successful update operation.
Function URL (api(us-central1)): https://us-central1-advertchain-firebase.cloudfunctions.net/api
i functions: cleaning up build files...
i hosting[advertchain-demo]: finalizing version...
+ hosting[advertchain-demo]: version finalized
i hosting[advertchain-demo]: releasing new version...
+ hosting[advertchain-demo]: release complete

+ Deploy complete!

Project Console: https://console.firebase.google.com/project/advertchain-firebase/overview
Hosting URL: https://advertchain-demo.web.app

c:\repos\advertchain-portal\nft-minter\functions(main -> origin)

```

Figure 33: *AdvertChain* Cloud functions deployed

7.2 Deploying the AdvertChain frontend

The *AdvertChain* NFT minter (frontend) requires the following libraries and third-party dependencies:

1. @alch/alchemy-web3@1.4.4
2. @google-cloud/secret-manager@4.0.0
3. cheerio@1.0.0-rc.12
4. ckey@1.0.3
5. cookie-parser@1.4.6
6. dotenv@16.0.1
7. find-config@1.0.0
8. firebase-admin@11.0.0
9. firebase-functions@3.22.0
10. firebase-tools@11.2.0
11. firebase@9.9.0
12. firebaseui@6.0.1
13. morgan@1.10.0
14. query-string@7.1.1
15. react-faq-component@1.3.4
16. request@2.88.2
17. superagent@7.1.6

To install these dependencies, go to the **nft-minter** folder and run the following command:

```
npm install -- save
```

Before deploying the frontend to Google's Firebase Cloud, we must first create a **.env** file with the environmental variables that the backend requires. Specifically, the following environmental variables are needed:

1. **REACT_APP_PINATA_KEY**, which is the API key for Pinata.
2. **REACT_APP_PINATA_SECRET**, which is the API secret for Pinata.
3. **REACT_APP_ALCHEMY_KEY**, which is the Alchemy HTTPS URL that contains the API key for the Mumbai testnet.
4. **REACT_APP_REDIRECT_URI**, which is the public URL of the serverless function that handles the LinkedIn Identity Federation redirect callback. This is the same URL that we configured in the *AdvertChain* developer app on LinkedIn in section 5.

5. **REACT_APP_CLIENT_ID**, this is the client id of the *AdvertChain* developer app on LinkedIn.
6. **REACT_APP_CLIENT_SECRET**, this is the client secret of the *AdvertChain* developer app on LinkedIn.
7. **REACT_APP_ADVERTCHAIN_CONTRACT**, which is the Mumbai testnet address of the deployed *AdvertChain* ERC-721 contract, as deployed in chapter 6.
8. **REACT_APP_WEB3_PUBLIC_KEY**, which is the public address (key) of the MetaMask wallet.
9. **REACT_APP_WEB3_PRIVATE_KEY**, which is the private key of the MetaMask wallet. Extract this key from MetaMask.
10. **REACT_APP_FIREBASE_API_KEY**, which is the API key for Firebase, as shown in the Firebase App Initialization Code in subsection 4.2.
11. **REACT_APP_FIREBASE_AUTH_DOMAIN**, which is the Firebase auth domain, as shown in the Firebase App Initialization Code in subsection 4.2.
12. **REACT_APP_FIREBASE_PROJECT_ID**, which is the Firebase project id, as shown in the Firebase App Initialization Code in subsection 4.2.
13. **REACT_APP_FIREBASE_APP_ID**, which is the Firebase app id, as shown in the Firebase App Initialization Code in subsection 4.2.

Figure 34 shows the environmental variables defined in the PoC `.env` file for the frontend.

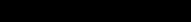
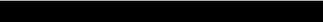
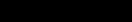
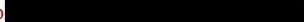
```
nft-minter > client >  .env
1  REACT_APP_PINATA_KEY = 2c21c
2  REACT_APP_PINATA_SECRET = d1c00e082c2e3d81ed
3  REACT_APP_ALCHEMY_KEY = https://polygon-mumbai.g.alchemy.com/v2/qXXMWG
4
5  REACT_APP_REDIRECT_URI = https://advertchain-demo.web.app/callback
6  REACT_APP_CLIENT_ID = 86561pedo8f9be
7  REACT_APP_CLIENT_SECRET = s03JhQv
8
9  REACT_APP_ADVERTCHAIN_CONTRACT = "0xC3a2eb67D1d0Df8bc49Aaf5aA4EcB540Ca4A7D5F"
10 REACT_APP_WEB3_PUBLIC_KEY = "0x538Ae2c94cbD9903709b5E0E5f4d2B4aB34Ff3eD"
11 REACT_APP_WEB3_PRIVATE_KEY = "0x4700f15f34061e"
12
13 REACT_APP_FIREBASE_API_KEY = "AIzaSyD1T6o1XUb"
14 REACT_APP_FIREBASE_AUTH_DOMAIN = "advertchain-firebase.firebaseio.com"
15 REACT_APP_FIREBASE_PROJECT_ID = "advertchain-firebase"
16 REACT_APP_FIREBASE_APP_ID = "1:1077202671127:web:257dd1
```

Figure 34: *AdvertChain* frontend environmental variables

Notice that all the environmental variables for the frontend start with the prefix **REACT_APP_**. This is required for the React.js app to identify and load these environmental variables. Also note that most of these environmental variables are common with the backend, with the difference that for the frontend they have slightly different names.

Before deploying the frontend, we must first build it. It is advisable to rebuild it every time there is any code change and before the code is deployed. To achieve this, enter the following command:

```
npm run build
```

To test the frontend locally, before deploying to the Firebase Cloud, run the following command from the **functions** folder:

```
firebase serve
```

If there are no compilation or runtime issues, *AdvertChain* will be deployed on localhost. Then, proceed to deploy the *AdvertChain* Cloud functions on Firebase by entering the following command:

```
firebase deploy
```

Figure 35 shows the Firebase frontend deployment commands in the PoC's environment.

```
c:\repos\advertchain-portal\nft-minter\client(main -> origin)
λ firebase deploy

=== Deploying to 'advertchain-firebase'...

i  deploying functions, hosting
i  functions: ensuring required API cloudfunctions.googleapis.com is enabled...
i  functions: ensuring required API cloudbuild.googleapis.com is enabled...
i  artifactregistry: ensuring required API artifactregistry.googleapis.com is enabled...
+  functions: required API cloudfunctions.googleapis.com is enabled
+  artifactregistry: required API artifactregistry.googleapis.com is enabled
+  functions: required API cloudbuild.googleapis.com is enabled
i  functions: preparing codebase default for deployment
i  functions: Loaded environment variables from .env.
i  functions: preparing functions directory for uploading...
i  functions: packaged c:\repos\advertchain-portal\nft-minter\functions (378.75 KB) for uploading
+  functions: functions folder uploaded successfully
i  hosting[advertchain-demo]: beginning deploy...
i  hosting[advertchain-demo]: found 34 files in client/build
+  hosting[advertchain-demo]: file upload complete
i  functions: updating Node.js 16 function api(us-central1)...
+  functions[api(us-central1)] Successful update operation.
Function URL (api(us-central1)): https://us-central1-advertchain-firebase.cloudfunctions.net/api
i  functions: cleaning up build files...
i  hosting[advertchain-demo]: finalizing version...
+  hosting[advertchain-demo]: version finalized
i  hosting[advertchain-demo]: releasing new version...
+  hosting[advertchain-demo]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/advertchain-firebase/overview
Hosting URL: https://advertchain-demo.web.app
```

Figure 35: *AdvertChain* Frontend deployed

As it can be seen from the figure, when the deployment to Firebase completes, the URL of the frontend is displayed. In our case, the hosting URL for the *AdvertChain* PoC frontend is: <https://advertchain-demo.web.app/>.

8 Running a Demo of the Proof-of-Concept

8.1 Prerequisites

At this point, it is assumed that all the following have been set up and deployed to the Cloud:

1. The public domain and website of the fake company (*Grecho*) has been created and deployed
2. The fake company's (*Grecho*) VC issuer has been configured and deployed
3. The Azure Verified ID (Verifiable Credentials) service has been configured
4. The Firebase project and Cloud services have been configured
5. The Pinata Cloud has been configured for IPFS
6. The Alchemy Web3 platform has been configured for the Mumbai testnet
7. The NoCode API platform has been configured for screenshot capturing
8. The LinkedIn OAuth Identity Federation has been configured
9. The *AdvertChain* ERC-721 contract has been deployed on the Mumbai testnet
10. The *AdvertChain* backend and frontend have been configured accordingly and deployed on the Firebase Cloud

If all steps have been completed successfully, we are ready to use the *AdvertChain* Proof-of-Concept solution. If not, go through the previous sections of this configuration manual on how to complete these steps.

8.2 User Stories

Before we describe the steps on how to use the *AdvertChain* Proof-of-Concept, we must first understand the solution's user stories.

There are two main actors (user types) of the system:

1. The job poster, who is an employee of the hiring company, authorized to create new job posts and *AdvertChain* NFTs
2. The job applicant, who wishes to verify the authenticity of a job post before he/she applies to the job

Following are the main user stories of the *AdvertChain* PoC solution:

1. As a job poster I want to login via LinkedIn, so that I am authenticated.
2. As a job poster I want to validate my identity via my VC, so that I am authorized.
3. As a job poster I want to connect my MetaMask wallet, so that I can mint NFTs.
4. As a job poster I want to be able to mint a LinkedIn job post as NFT, so that I can prove the ownership of the job post to job applicants.
5. As a hiring company I want to authorize only my Hiring Managers and Recruiters to create job posts and mint *AdvertChain* NFTs.

6. As a job applicant I want to verify a job post, so that I know its ownership and identity are authentic before I apply.

The first five user stories apply to the job poster user. The last user story applies to the job applicant user.

8.3 The job poster user journeys

Before the job poster begins his/her user journey, the MetaMask browser plugin must be installed on his/her browser and test Matic cryptocurrency must have been added to his/her Mumbai wallet. Also, the job poster must have the latest version of the Microsoft Authenticator app installed on his/her phone.

8.3.1 Issuing an employee Verifiable Credential to the job poster

We assume that the job poster is an employee of the *Grecho* company with the job role of a **recruiter**.

Before the job poster uses the *AdvertChain* solution to mint job posts as NFTs, the job poster must receive a Verifiable Credential (VC) from *Grecho* (the hiring company he/she works for) that verifies that the job poster is a verified *Grecho* employee whose job role is **recruiter**. The VC will also contain the employee's full name and email.

The issuance of the employee Verifiable Credential (VC) is a once-off step. It must be done only once and the VC remains valid until the issuing company (*Grecho*) revokes it or unless the user deletes the VC from his/her phone device.

To receive a VC from *Grecho*, go to <https://grecho-vc-issuer.azurewebsites.net/> and click the "**Get Credential**" button. A QR code will be displayed, as shown in the Figure 36. Scan the QR code with the Microsoft Authenticator app and enter the pin which is displayed under the QR code. At that point the QR code will be hidden and the message "QR Code is scanned. Waiting for issuance to complete..." will be displayed, as shown in the Figure 37. After a while, a screen will appear on the Microsoft Authenticator app, requesting the user to accept the new Verifiable Credential. After the user accepts the newly issued Verifiable Credential, the Verifiable Credential will be stored in the Microsoft Authenticator app that will act as a VC wallet for the user.

Finally, when the VC has been issued, the user can see the details and the claims of the newly issued VC, as shown in the Figure 38.

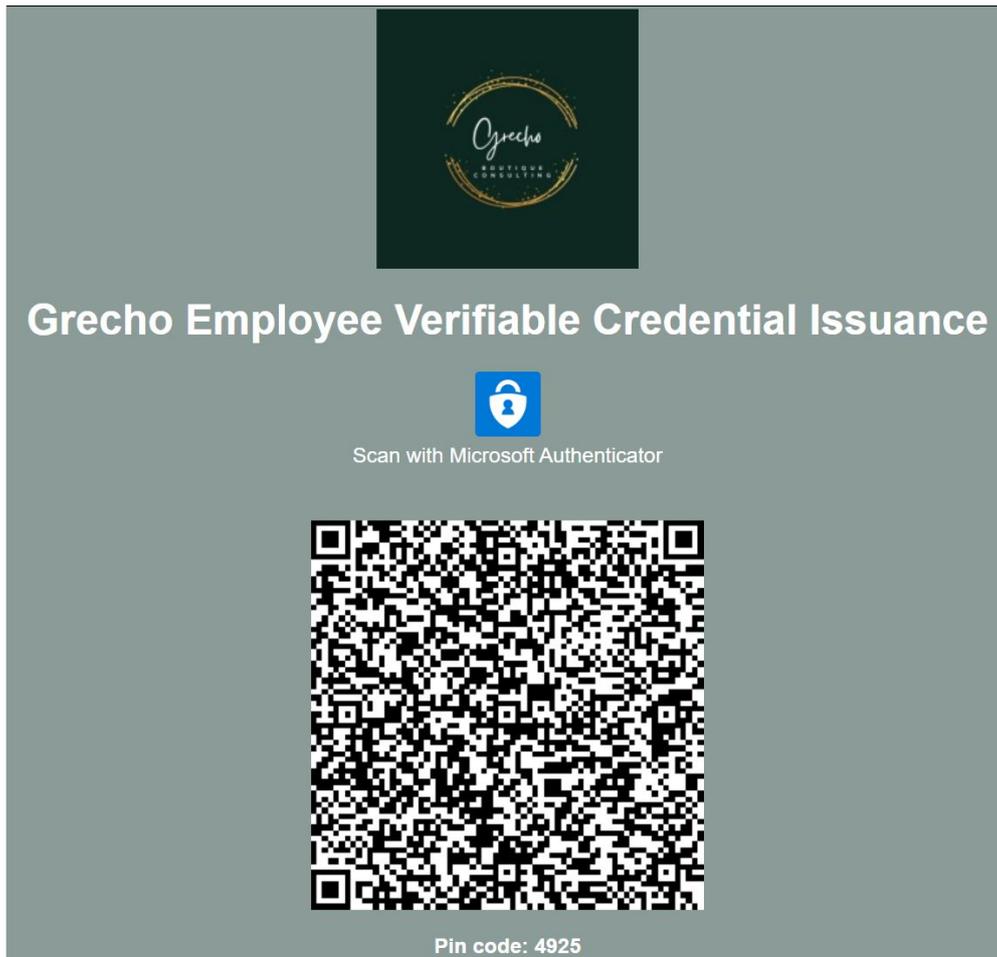


Figure 36: Verifiable Credential Issuance

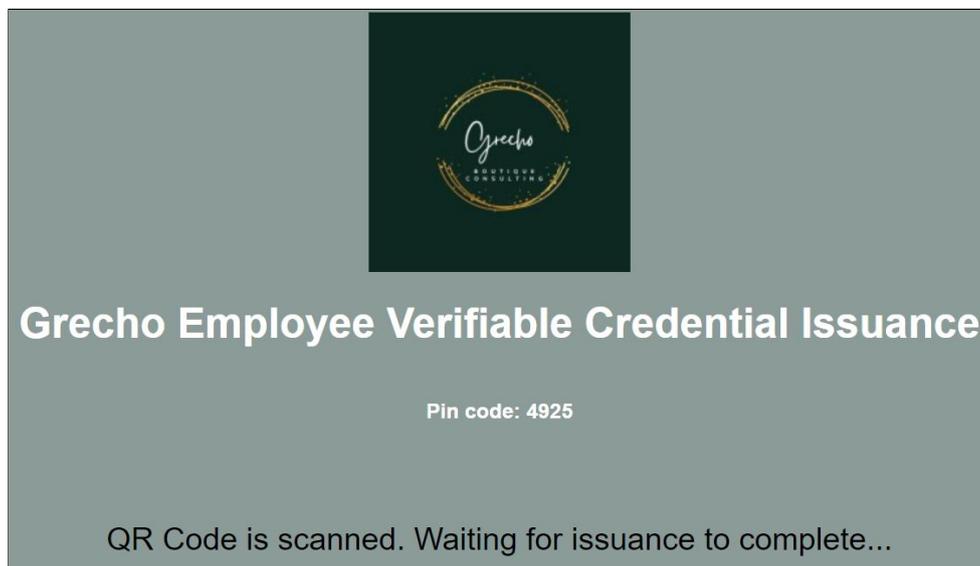


Figure 37: Waiting for Verifiable Credential Issuance

8.3.2 Minting job posts as *AdvertChain* NFTs

At this point, the user (job poster) has a valid Verifiable Credential (VC) on his/her Microsoft Authenticator app on his/her phone that proves he/she is a verified employee

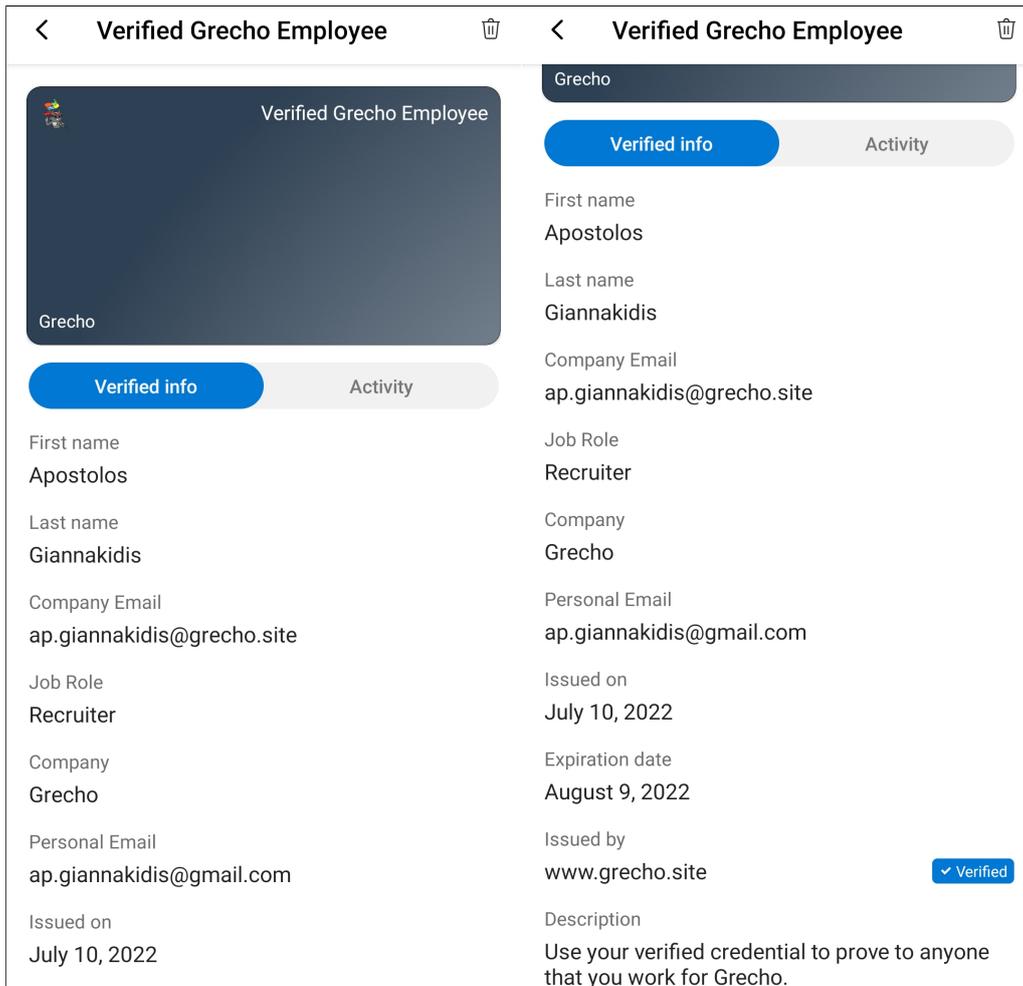


Figure 38: Verified Grecho Employee VC Claims

of *Grecho*. These VC claims will be used to authorize the user to mint *AdvertChain* NFTs.

It is now time to start the NFT minting process using the *AdvertChain* NFT minter. We assume that the job poster has already created the job post on LinkedIn and has the URL of the job post.

First, visit the *AdvertChain* NFT minter homepage. In our Proof-of-Concept, the *AdvertChain* NFT minter is accessible from:

<https://advertchain-demo.web.app/>

The screenshot in Figure 39 shows the *AdvertChain* NFT minter homepage, before any further user action.

Notice that the only user input elements are the three (3) blue/white buttons at the top of the page and one (1) text box around the middle of the page.

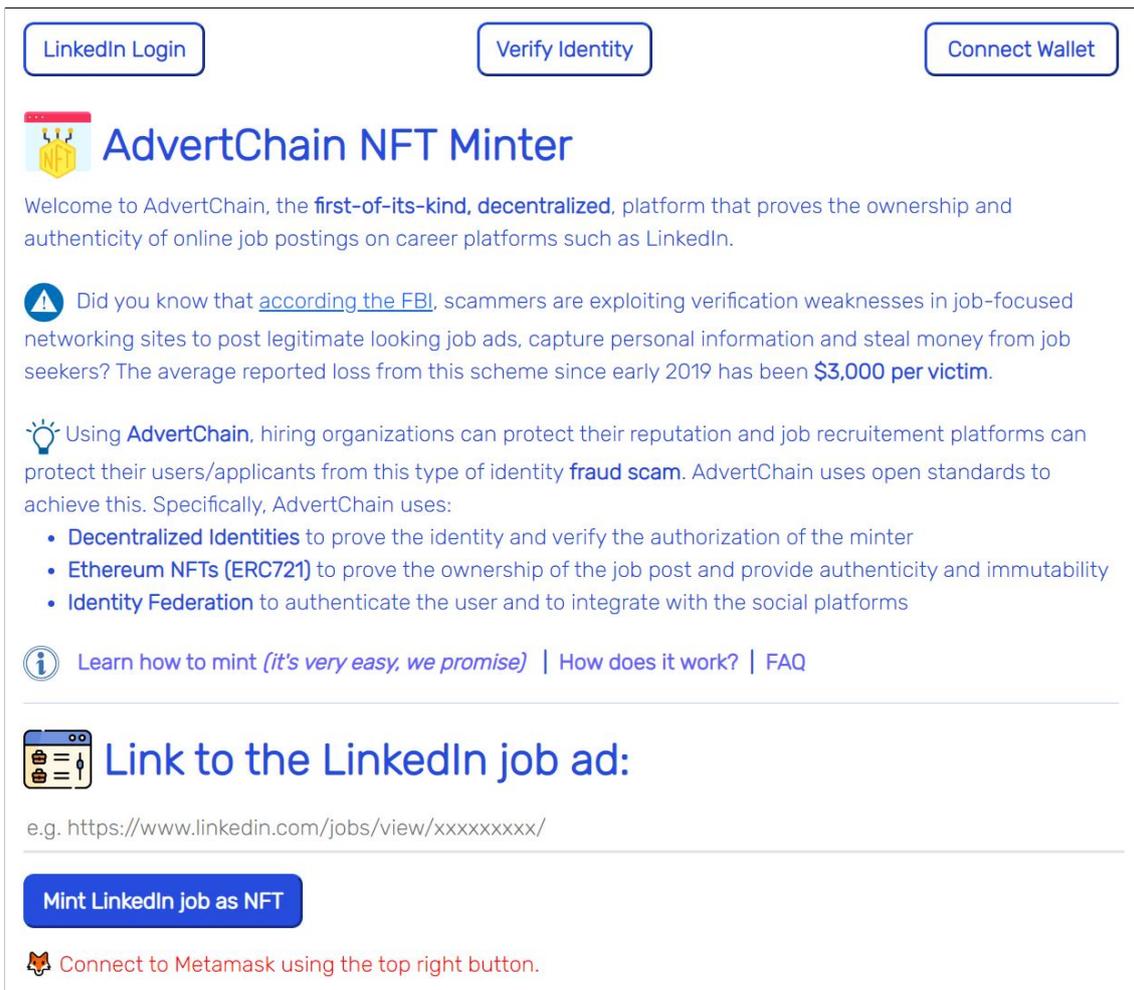


Figure 39: *AdvertChain* NFT minter homepage before any user action

The blue/white buttons at the top of the page are the following:

1. **LinkedIn Login** button that handles the Identity Federation
2. **Verify Identity** button that handles the user's VC verification
3. **Connect Wallet** button that handles the connection with MetaMask

The text box at the middle of the page has the title "**Link to the LinkedIn job ad**". Users paste the LinkedIn job post in this text box for the job post that should be minted as NFT. Under the text box there is a single blue button that initiates the NFT minting process.

Thus, users must perform four (4) simple steps that correspond to the four (4) input elements mentioned above. Also, these four (4) steps correspond to the user stories 1, 2, 3 and 4.

The user story 5 corresponds to the authorization business logic that is handled when the "**Mint LinkedIn job as NFT**" blue button is selected by the job poster, who is a verified/authorized employee of the hiring company.

The screenshot in Figure 40 highlights the UI input elements of the *AdvertChain* NFT minter homepage.

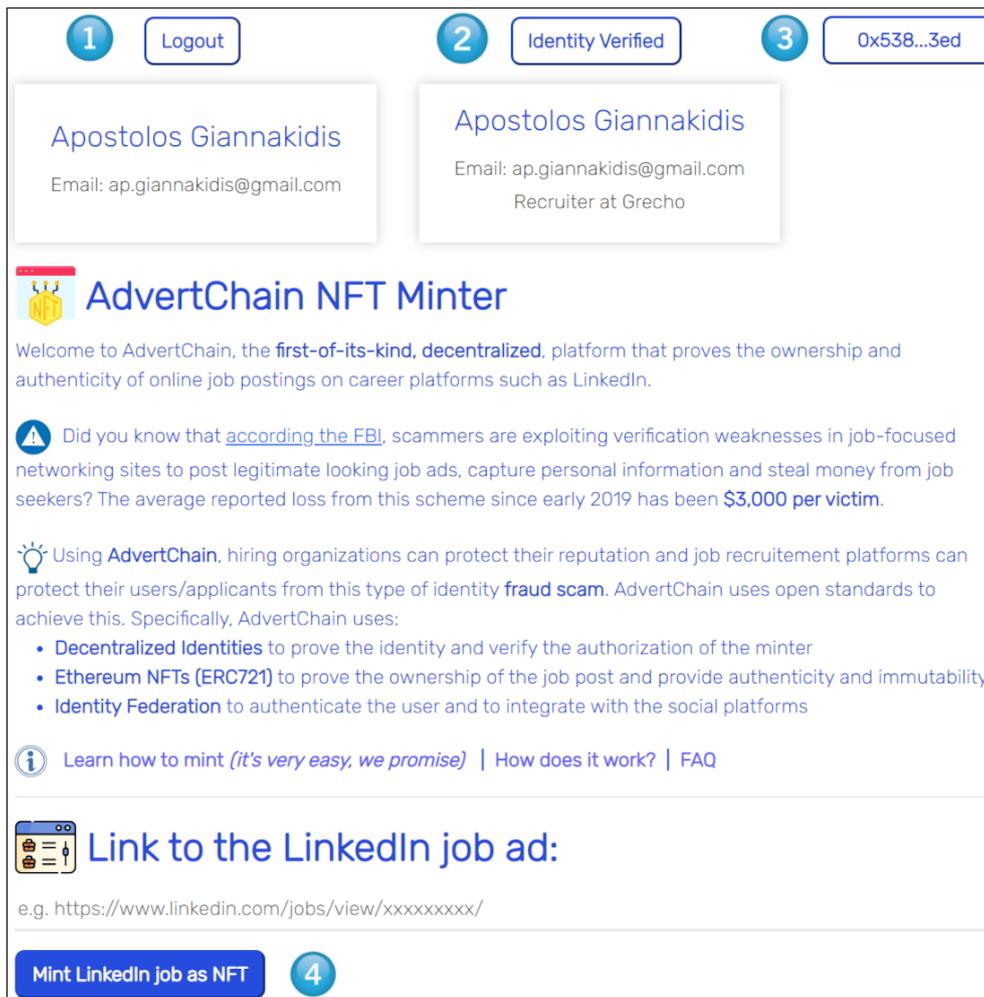


Figure 40: Highlighted input UI elements of the *AdvertChain* NFT minter

Next we will describe the user journeys of the five (5) user stories of the job poster:

User Story #1 As a job poster I want to login via LinkedIn, so that I am authenticated

The first step of the user journey is to login via LinkedIn. Click on the **LinkedIn Login** button. A popup window will appear that shows a login form provided by linkedin.com. Users can safely login because their user credentials will be sent directly to LinkedIn. This login form is shown in the Figure 41.

After providing the user credentials and clicking the "**Sign in**" button, the screen shown in Figure 42 will appear that requests the user's permission to share his/her LinkedIn profile details (full name and email) with *AdvertChain*. After the user allows *AdvertChain* to access the user's LinkedIn profile, *AdvertChain* will display the user's full name and email in a box under the **LinkedIn Login** button.

User Story #2 As a job poster I want to validate my identity via my VC, so that I am authorized

Next, the user must validate his/her identity via his/her Verifiable Credential. *AdvertChain* has integrated with the Azure Verified ID service and has the capability to verify

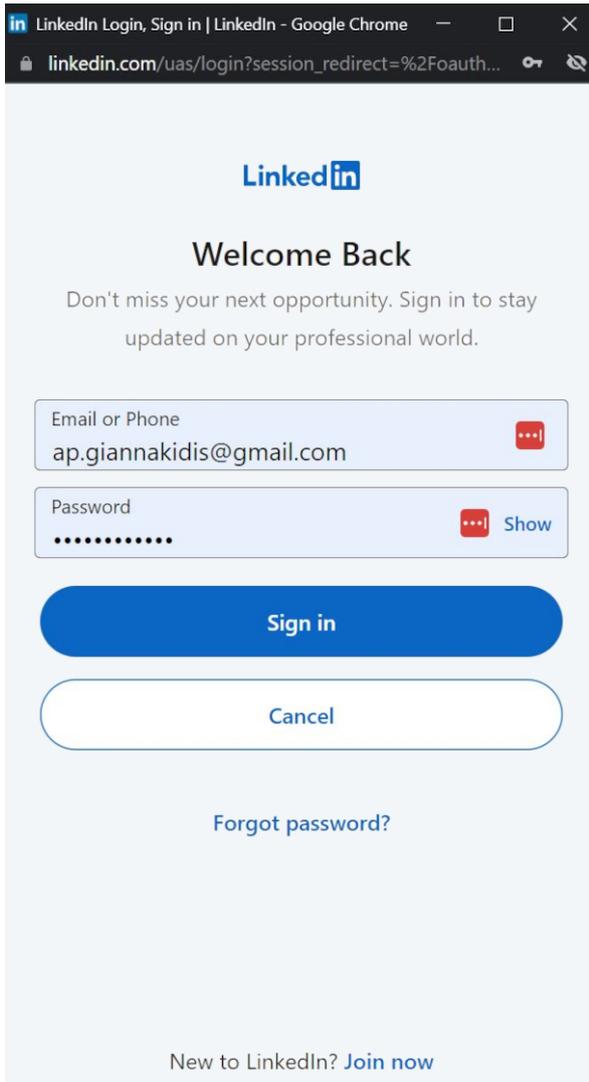


Figure 41: LinkedIn Federated Login

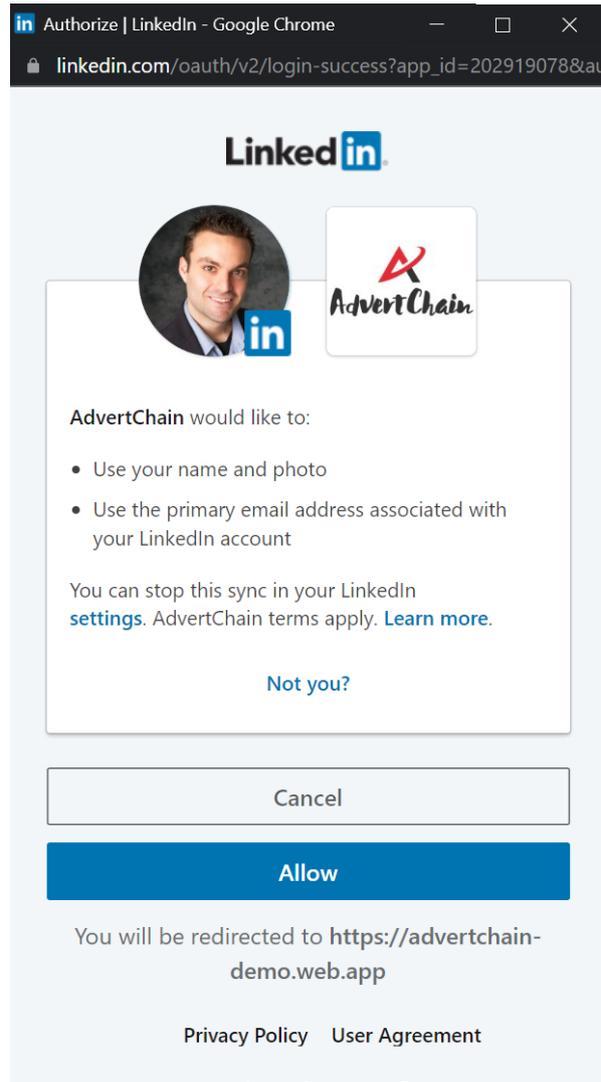


Figure 42: LinkedIn Authorization

presented Verifiable Credentials. To do this, click on the **Verify Identity** button at the top. This will display a pop up window that contains a single button, as shown in Figure 43. After click the **Verify Credential** button, a QR code will be displayed. Scan the QR code with you Microsoft Authenticator app. After a while, the mobile app will request your permission to share the VC with *AdvertChain*. Select **Allow** to continue. At that point, *AdvertChain* will receive the claims contained in the user's Verifiable Credential and it will display the user's full name and email in a box under the **Verify Identity** button, as shown in Figure 44

User Story #3 As a job poster I want to connect my MetaMask wallet, so that I can mint NFTs

The next step is to connect the user's MetaMask wallet with *AdvertChain*. This is a trivial step to perform. By selecting the **Connect Wallet** button, the user will be prompted by MetaMask to authorize the MetaMask wallet connection with *AdvertChain*. This is shown in Figure 45. After clicking Next, *AdvertChain* will connect with the user's MetaMask wallet and the account's public address will be shown in the button's label.



Figure 43: Verifying a VC in *AdvertChain*

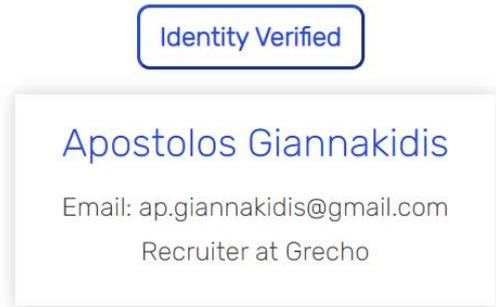


Figure 44: Verified employee's VC

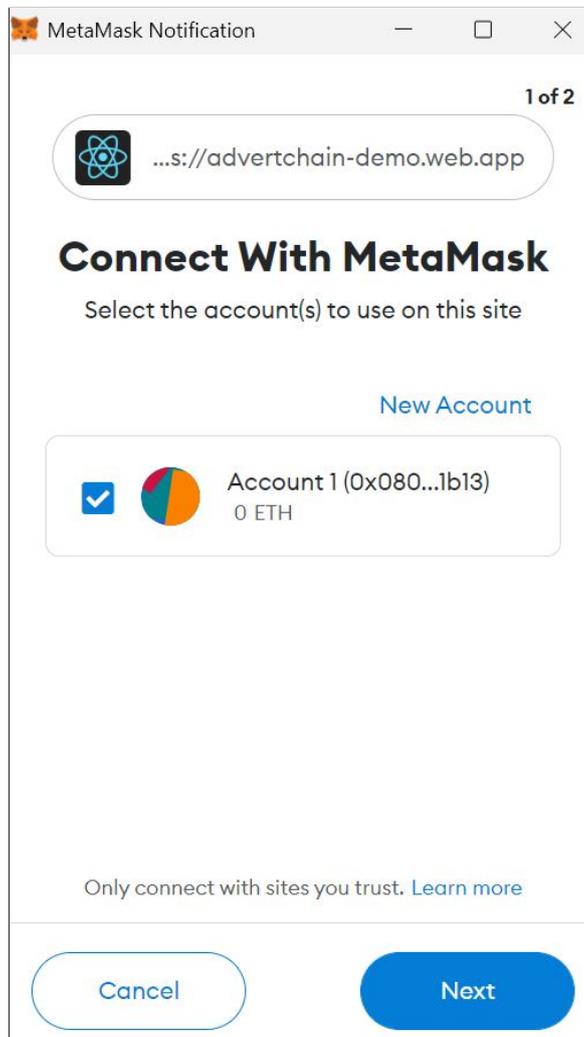


Figure 45: Authorize MetaMask to connect with *AdvertChain*

User Story #4 As a job poster I want to be able to mint a LinkedIn job post as NFT, so that I can prove the ownership of the job post to job applicants

At this point, *AdvertChain* has all the identity information it needs to enable the NFT minting functionality. The user (job poster) should now identify an active LinkedIn job post created by his employer, copy paste the job post's URL into the *AdvertChain* text box and then click the "Mint LinkedIn job as NFT" blue button. After clicking this button, the user should wait until the NFT has been minted. Typically, this process takes about one (1) minute to complete but it may take longer depending on the load of the Polygon network. After the NFT minting process has completed, the new NFTs Blockchain address will be displayed as well as a link to see the NFT on OpenSea. Figure 46 shows the output after a successful minting of an *AdvertChain* NFT.



Figure 46: Successful minting of an *AdvertChain* NFT

Users can click the displayed Polygonscan link to see more information about the Blockchain transaction on the Polygon network for the NFT minting. The second link goes to **OpenSea** so that the user can see the minted NFT in a visual way. For more information about OpenSea, go to subsection 8.5.

For the advanced users and for troubleshooting, the Javascript console provides high-level information about the steps that *AdvertChain* performs to mint new NFTs. Open the Javascript console on your browser and then mint a new a NFT. You should see console messages such as the ones shown in Figure 47.

At it can be seen in the Figure 47, *AdvertChain* first identifies the LinkedIn job post id and then checks if an *AdvertChain* NFT already exists for that id. It then starts the process of creating the NFT metadata that will be sent to Pinata. After creating the NFT metadata, *AdvertChain* signs the metadata, creates a screenshot of the job post and pins the screenshot and the metadata to IPFS via Pinata. Then, a new NFT is minted on the *AdvertChain* contract whose token URI points to the newly created NFT metadata on IPFS. After the NFT minting transaction is initiated, the transaction hash is printed. After the NFT minting finishes, the message "Successfully minted new NFT with Token ID:" is displayed.

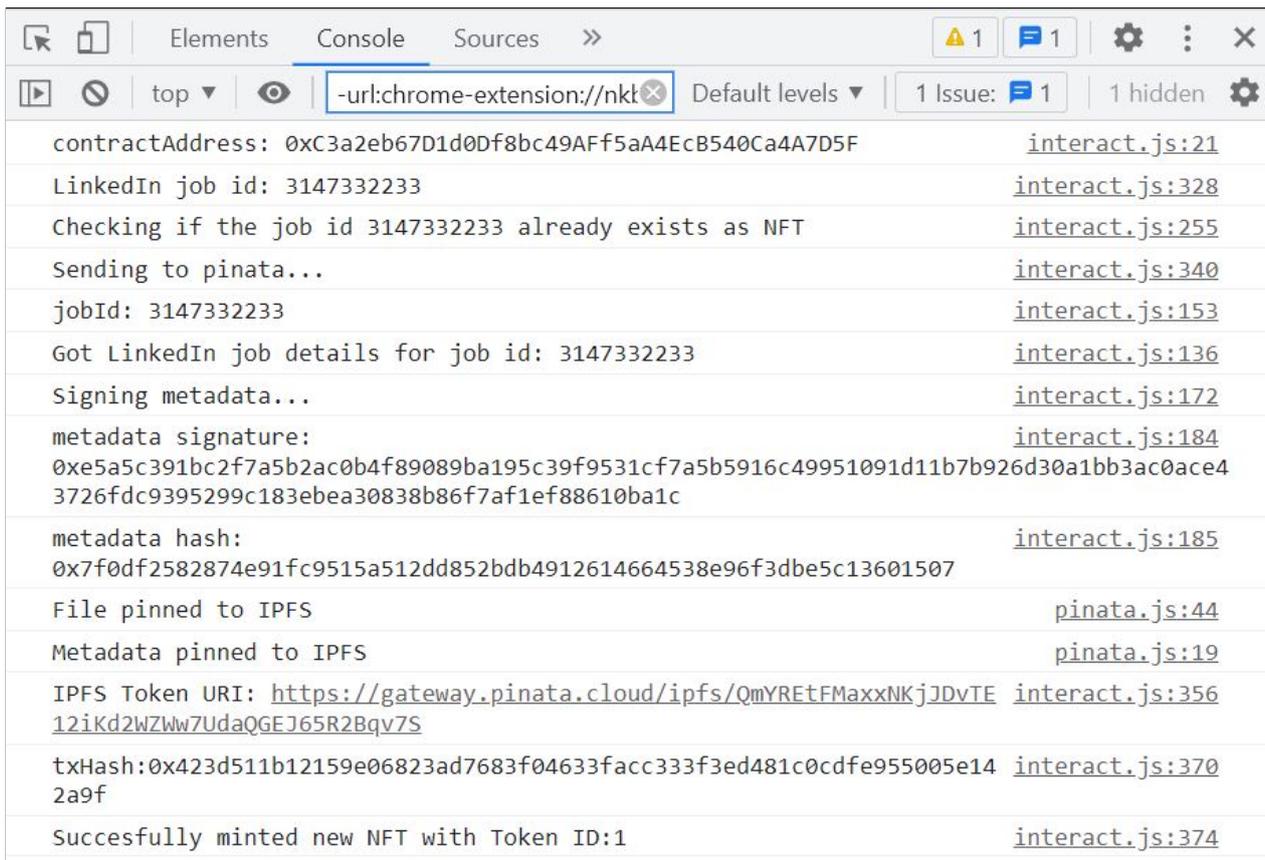


Figure 47: JS console messages during the minting of a new *AdvertChain* NFT

User Story #5 As a hiring company I want to authorize only my Hiring Managers and Recruiters to create job posts and mint *AdvertChain* NFTs

When the user pastes a LinkedIn job post and then clicks the "Mint LinkedIn job as NFT" blue button, *AdvertChain* performs a sets of authorization checks. Specifically, the following authorization checks are performed:

1. The full name from LinkedIn must match the full name from the Verifiable Credential (VC)
2. The user email from LinkedIn must match one of the emails contained in the VC claims
3. The VC must contain a claim that proves your job role allows you to post job ads. Typically, this means your job role is Recruiter or Hiring Manager.
4. The company name of the LinkedIn job ad must match the company name in the VC claims.

Thus, the LinkedIn job post that the job poster will paste into the *AdvertChain* NFT minter app must have been created by the same company that the job poster works for and that the job poster is a Recruiter or Hiring Manager verified by the hiring company via the issued VC.

If the user (job poster) is not authorized to mint *AdvertChain* NFTs, the *AdvertChain* NFT minter app will notify the user via an error message. An example error message

that notifies the user that he/she is not authorized to mint *AdvertChain* NFTs is shown in Figure 48. In this specific example, the user is not authorized to mint *AdvertChain* NFTs due to the fact that his/her federated LinkedIn name is different than the user's name in the claims of the Verifiable Credential issued by his/her employer.



Figure 48: Unauthorized user cannot mint *AdvertChain* NFTs

8.4 The job applicant user journey

From the job applicant's point of view, the user story is different. The job applicant does not mint NFTs. The job applicant only wants to verify the authenticity of LinkedIn job posts. More specifically, the job applicant should verify the job posts's ownership and the job poster's identity before the job applicant applies to the job post.

The *AdvertChain* backend exposes a REST API to allow for programmatic verification of a job post. This REST API is accessible via the REST API endpoint <https://us-central1-advertchain-firebase.cloudfunctions.net/api/validateJobPost> that accepts the URL of the LinkedIn job post via the *jobUrl* GET query parameter. For example, to verify the job post with the URL <https://www.linkedin.com/jobs/view/3147332233/>, then a GET request must be sent to: <https://us-central1-advertchain-firebase.cloudfunctions.net/api/validateJobPost?jobUrl=https://www.linkedin.com/jobs/view/3147332233/>. Because this job post has already been minted as an *AdvertChain* NFT, the response of this HTTP REST call will be:

```
{"verifiedOwnership":true,"verifiedIdentity":true}
```

The job applicant does not require a MetaMask wallet nor a Verifiable Credential to verify a job post via *AdvertChain*.

To make *AdvertChain* easier to use for the job applicants, a Tampermonkey userscript has been developed.

Tampermonkey is a very popular browser extension that is used to run so-called userscripts (sometimes also called Greasemonkey scripts) on websites. Userscripts are small computer programs that change the layout of a page, add or remove new functionality and content, or automate actions.

The *AdvertChain* userscript automates the REST API call to the *AdvertChain* job post verifier and displays a green check on the job post, if the job post can be successfully verified by *AdvertChain*.

Make sure to install Tampermonkey from your browser's extensions page, for example: <https://chrome.google.com/webstore/detail/tampermonkey/dhdgffkkehbmkfjojejmpbldmpobfk>

hl=en. Then, install the *AdvertChain* userscript which is available from: <https://github.com/maestros/advertchain-portal/tree/main/userscript>.

Due to a Content-Security-Policy restriction, we will also have to install the **Disable Content-Security-Policy** browser extension. This can be installed from:

<https://chrome.google.com/webstore/detail/disable-content-security/ieelmcmcagomplcee>
hl=en.

After enabling the Disable Content-Security-Policy extension and the *AdvertChain* Tampermonkey userscript, then go to a LinkedIn job post page, for example: <https://www.linkedin.com/jobs/view/3147332233/>.

Figure 49 shows how the Tampermonkey userscript works for an example LinkedIn job post that has been successfully validated by *AdvertChain*.

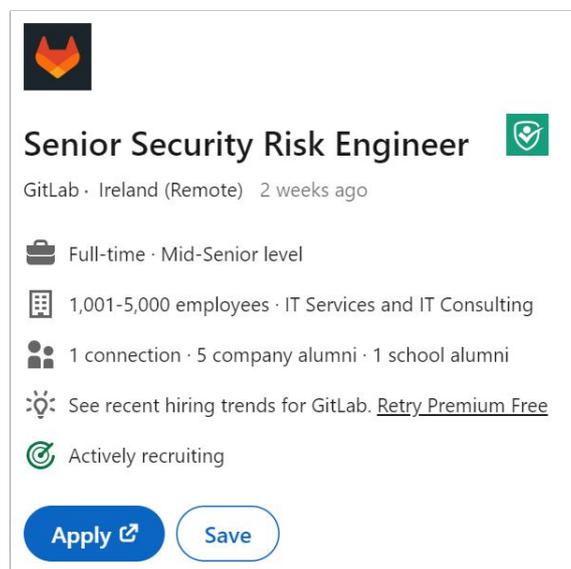


Figure 49: Successfully validated LinkedIn job post via *AdvertChain*

Figure 50 shows the code of the *AdvertChain* Tampermonkey userscript.

```
25 var url = 'https://us-central1-advertchain-firebase.cloudfunctions.net/api/validateJobPost?jobUrl=' + window.location.href;
26
27 $(document).ready(function(){
28   $.ajax({
29     type: 'GET',
30     url: url,
31     dataType: 'json'
32   })
33   .done(function(data, textStatus, responseObject) {
34     var jsonData = responseObject.responseJSON;
35     var verifiedOwnership = jsonData.verifiedOwnership;
36     var verifiedIdentity = jsonData.verifiedIdentity;
37
38     var hoverText = 'The identity of the job poster has been verified. You can apply securely.';
39
40     var verifiedIcon = '&nbsp; &nbsp; ';
41
42     if (verifiedIdentity == true) {
43       $('<div>.t-24.t-bold.jobs-unified-top-card__job-title</div>').first().append(verifiedIcon);
44     }
45   })
46 })
47 })
```

Figure 50: *AdvertChain* Tampermonkey userscript

8.5 Browsing the *AdvertChain* NFTs

Currently, the *AdvertChain* Proof-of-Concept platform does not offer a native way to explore the collection of the minted NFTs in a graphical way. For this reason, we will use the functionality offered by **OpenSea**, the most popular NFT marketplace. OpenSea allows users to view collections of NFTs and provides tools to buy, sell and make offers on NFTs. In our use case, the *AdvertChain* does not allow NFTs to be traded, so OpenSea's buy/sell functionality is not applicable. We will only use OpenSea to be able to browse the *AdvertChain* NFT collection. Figure 51 shows an example *AdvertChain* NFT as displayed on OpenSea. The *AdvertChain* NFT collection can be accessed via OpenSea on: <https://testnets.opensea.io/collection/advertchain>.

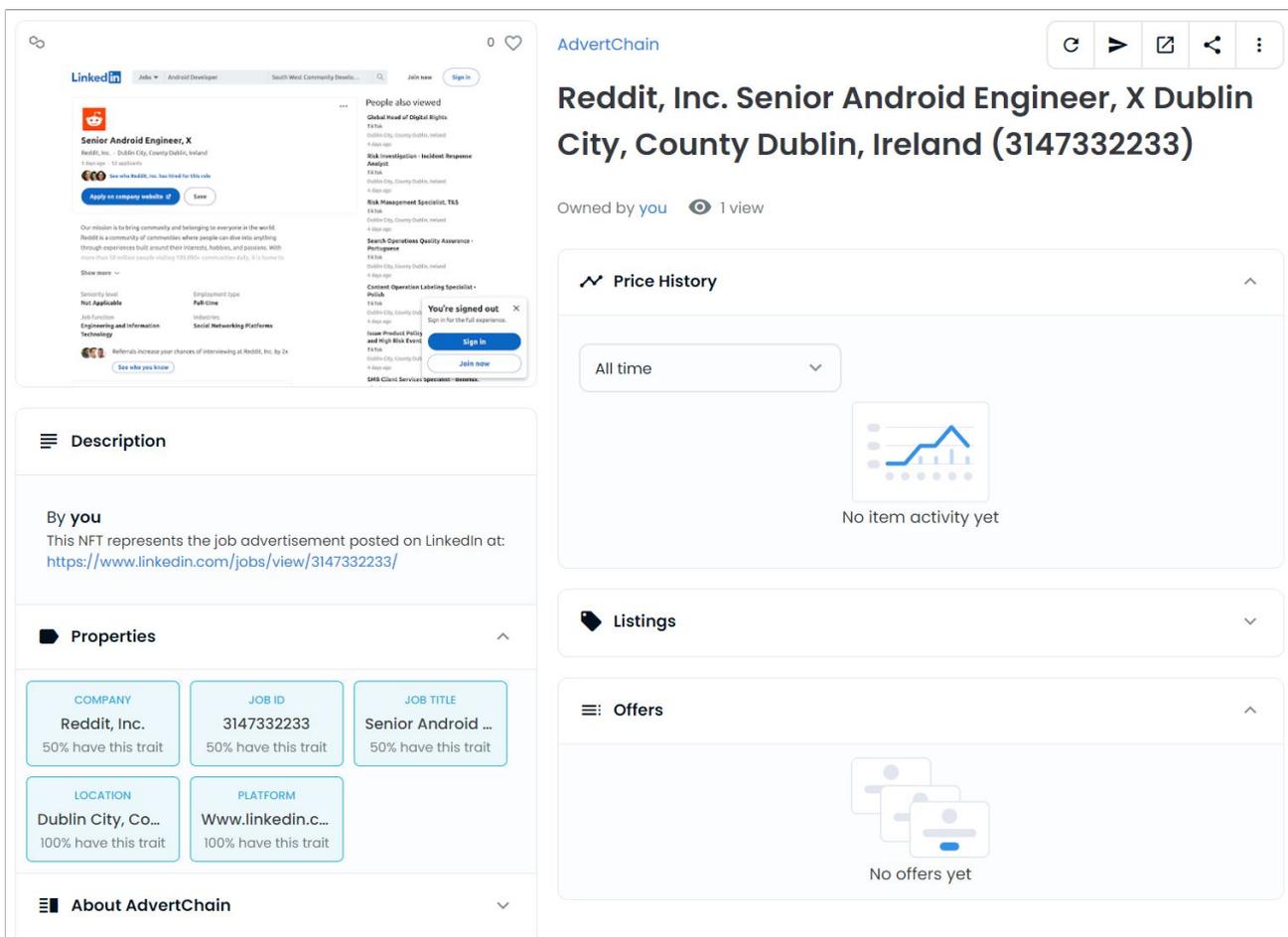


Figure 51: Sample *AdvertChain* NFT on OpenSea

9 Frequently Asked Questions

1. What is AdvertChain?

AdvertChain is a decentralized solution to the identity theft scam in job recruitment social platforms, as described by the FBI's public service announcement I-020122-PSA. The goal of AdvertChain is to protect applicants from not falling victims of this scam as well as to protect the reputation of the hiring companies whose brand name and identity is used to scam the victims.

2. How does AdvertChain work?

AdvertChain uses a combination of recent technologies and open standards to provide the security guarantees required to ascertain the authenticity, ownership and integrity of LinkedIn job posts. More specifically, AdvertChain uses NFTs to provide ownership and copyright of the job post, Decentralized Identity (DiD), also known as Self-Sovereign-identity, provides a secure and tamper-proof solution for identifying individuals and for access control. Identity Federation allows users to authenticate using their LinkedIn account and enables the integration with the LinkedIn platform. After a user has been identified, authenticated and authorized, they can mint a LinkedIn job post as an NFT on Polygon, a Layer 2 network that works atop the Ethereum blockchain. Each NFT contains metadata that links the NFT to the corresponding LinkedIn job post and the authenticated individual. Also, each NFT's metadata is cryptographically signed by AdvertChain, making sure that only the NFTs that are minted via the AdvertChain's platform are legitimate. When verifying a LinkedIn job post, AdvertChain checks if there is an NFT minted for that job post, if the NFT's metadata match and if the signature matches.

3. Why can't we trust the job recruitment platform (e.g. LinkedIn) to provide the same security guarantees for the jobs posted on their platform?

According to the FBI public service announcement I-020122-PSA, the scammers exploit security weaknesses on job recruitment websites to post fraudulent job postings. Those postings appear alongside legitimate jobs posted by the hiring companies, making it difficult for applicants and the spoofed company to discern which job posting was real and which one was fraudulent. Even if a job recruitment website fixes the identified security vulnerabilities, there is always the risk of the introduction of new security vulnerabilities or zero days. Therefore, the most secure approach is to use an external, decentralized, system to verify the authenticity and legitimacy of the job posts.

4. What does it mean that AdvertChain is a decentralized solution?

AdvertChain is primarily using decentralized technologies based on Blockchain, such as Decentralized Identifiers, Verifiable Credentials, and NFTs. The Decentralized Identity service is provided by Microsoft's Azure AD Verifiable Credentials, which is based on a Layer 2 open, permissionless, sidetree-based DID network, called ION (Identity Overlay Network) which runs atop the Bitcoin blockchain. Note that the Azure AD Verifiable Credentials service is still in a public preview release but it is expected to go GA soon. The AdvertChain NFTs are implemented using the Open Zeppelin ERC721 contract and it is deployed on the Polygon network. Due to the Proof-of-Concept nature of AdvertChain,

the NFTs are deployed on the Mumbai testnet and not on the Polygon mainnet. It must be noted that AdvertChain is also making use of public Clouds for its operation. Specifically, AdvertChain is deployed on Google's Firebase and it makes use of Firebase Functions and the Firestore database.

5. What are the Decentralized Identity (DiD) and Verifiable Credentials (VC) and why AdvertChain uses them?

Decentralised identity is an emerging concept that gives back control of identity to consumers through the use of an identity wallet in which they collect verified information about themselves from certified issuers. Verifiable Credentials are identity attestations that come from a trusted issuer, like proof of a workplace. In a decentralised identity framework, the individual receives verifiable credentials proving their identity from the trusted issuer and stores them in a digital wallet. The user (aka Holder) can then present proofs of their identity to any company that requests it (Verifier), and these companies can verify that the proofs are true via a Blockchain-based ledger. In this case, AdvertChain is the Verifier, the DiD Blockchain-based ledger is ION (Identity Overlay Network) and the digital wallet is the Microsoft Authenticator mobile app. Decentralized identifiers, are a standard developed by the World Wide Web Consortium (W3C). for decentralized identifiers and public key infrastructure. AdvertChain uses DiD and VCs to verify that the individual that wishes to mint a job post is in fact an employee of the hiring company, they are authorized to post job ads on career platforms and that individual's name and email match the ones on the career social platform (LinkedIn).

6. Do I need to create an account to use AdvertChain?

No, users do not need to create an AdvertChain account. AdvertChain does not have or require a sign-up functionality. However, users will require a LinkedIn account, a valid Verifiable Credential issued by a DiD authority that matches the name and email of the LinkedIn account, and a MetaMask wallet. It is important to understand that in order to use AdvertChain, the company that wishes to secure their LinkedIn job posts must issue Verifiable Credentials to their authorized employees. The issued Verifiable Credential must match the name of the LinkedIn account and must contain the employee's personal email in the claims.

7. Do I need a LinkedIn account to mint an AdvertChain NFT?

Yes, you will need a valid LinkedIn account. The email of that LinkedIn account must match the email contained in the claims of the Verifiable Credential issued by the company that has authorized you to post job posts on LinkedIn.

8. Is AdvertChain free to use?

Yes, AdvertChain is currently free to use. AdvertChain is in a Proof-of-Concept state and uses Mumbai, a Polygon test network that is free to use. The AdvertChain Proof-of-Concept was not created with profit in mind.

9. Who can mint LinkedIn job ads as NFTs?

Anyone can mint AdvertChain NFT as long as they fulfil the following requirements: 1.

The user must have a valid MetaMask wallet 2. The user has been successfully authenticated using his/her LinkedIn account 3. The user has a valid Verifiable Credential (VC) issued by the hiring company he/she is minting the NFT for 4. The user's job role is a Recruiter or a Hiring Manager, as stated in the claims of the VC. 5. The user's LinkedIn email matches the user's email from his VC. Users who fulfil the above requirements can mint LinkedIn jobs as AdvertChain NFTs.

10. Is AdvertChain a cryptocurrency?

No, AdvertChain is not a cryptocurrency. The AdvertChain ERC721 NFT smart contract does not set a price to the minted NFTs.

11. Who owns the minted NFTs?

The minted AdvertChain NFTs are owned by the address of the connected MetaMask wallet that was used to mint each NFT.

12. Can I sell my minted AdvertChain NFT?

No, you can't. The AdvertChain ERC721 smart contract has been designed in such a way that it does not allow a minted NFT to change ownership. Thus, AdvertChain NFTs cannot and should not be sold.

13. What are my minted AdvertChain NFTs worth?

There is no monetary value to the AdvertChain NFTs.

14. Should I invest in the AdvertChain NFTs?

The NFTs that are minted by AdvertChain are not meant to be used for profit or investment.

15. Why does AdvertChain use NFTs?

The AdvertChain NFTs are serving the purpose of tracking the ownership and copyrights of the job post that they are linked to.

16. Why does AdvertChain use MetaMask?

AdvertChain mints NFTs on the Polygon network, which is a Layer 2 network that works atop the Ethereum blockchain. Every transaction on the blockchain must be signed by a valid Ethereum client. The MetaMask wallets is used to sign transactions on behalf of its owner. Also, the minted NFT will be owned by the address of the connected MetaMask wallet.

17. I'm the owner of a LinkedIn job post and I modified it. Can I modify the NFT?

NFTs cannot be modified. Once an NFT is minted on the blockchain, it is considered immutable. This means it cannot change and it will exist on the blockchain network forever. In this case, create a new LinkedIn job and mint a new AdvertChain NFT.

18. Can I delete an AdvertChain NFT?

NFTs cannot be deleted. Once an NFT is minted on the blockchain, it is considered immutable. This means it cannot change and it will exist on the blockchain network forever.

19. Is AdvertChain affiliated with LinkedIn?

AdvertChain is not affiliated with LinkedIn in any professional, financial or contractual way.

20. Why was LinkedIn chosen?

Currently, AdvertChain is a Proof of Concept (PoC). It aims to prove that the main idea of using NFTs + DIDs + Identity Federation can provide a working solution to the identity theft scam in job recruitment social platforms. For the PoC to work, a job recruitment social platform had to be used. LinkedIn was chosen as the most popular social platform in the domain of job recruitment / career networking.

21. Is AdvertChain using, handling or storing any user sensitive information?

AdvertChain does not store any personal or user sensitive information. Every type of information that AdvertChain uses is publicly available. The only personal piece of information are the claims provided by the Verifiable Credential. However, it must be highlighted that the user gives explicit consent to share these claims with AdvertChain. Importantly, AdvertChain does not store these claims in any database, apart from the cryptographic hash of the DiD.

22. Is there an API that I can use to verify LinkedIn job posts programmatically?

Yes, there is: <https://us-central1-advertchain-firebase.cloudfunctions.net/api/validateJobPost>. This API endpoint accepts a single query parameter named **jobUrl** whose value is the URL of the LinkedIn job post. This API endpoint returns a JSON string with two fields indicating whether the ownership and the identity of the minter have been verified. For example: "verifiedOwnership":true,"verifiedIdentity":true

23. Is there a more user friendly way to verify LinkedIn job posts?

Yes, there is. AdvertChain provides a Tampermonkey userscript that automates the verification process. Simply, install Tampermonkey plugin in your browser (<https://chrome.google.com/webstore/detail/tampermonkey/dhdgffkkehbmkfjojejmpbldmpobfkfo?hl=en>) and install the AdvertChain userscript. Then visit the LinkedIn job post you wish to verify. If the job post is verified by AdvertChain, you will see a green check next to the job post title.

24. I am using the AdvertChain Tampermonkey userscript for LinkedIn and I don't see a green check next to a job post. What does this mean?

It means that there is no AdvertChain NFT for the corresponding LinkedIn job post. AdvertChain is able to verify the authenticity and legitimacy of job posts that have been

minted as NFTs. It is important to understand that a lack of AdvertChain verification does not mean that the LinkedIn job post is fraudulent. It only means that AdvertChain cannot verify its authenticity and legitimacy.

25. Who created AdvertChain?

Apostolos Giannakidis came up with the idea of AdvertChain as a solution to the identity fraud scheme as described by the FBI's public service announcement I-020122-PSA that was released on February 01, 2022.

26. What is the current state of AdvertChain?

The current version of AdvertChain is a Proof of Concept that aims to demonstrate how the proposed main ideas and concepts can work together and provide a solution to the identity fraud that exploit job advertisements in social networks.

27. What is the future state of AdvertChain?

There are no concrete plans for future expansion, however if this Proof of Concept proves to be successful, then there will be efforts to improve AdvertChain and add more features.

28. Is AdvertChain an open source software?

AdvertChain is free to use but its source code of AdvertChain has not been open sourced yet. However, there are plans to open source the source code repository in the future. If you wish to contribute to the project watch this space or reach out to ap.giannakidis@gmail.com