MSDF SDCOE Ed-Fi Transcript Prototype: Windows Deployment Guide

Revision July 2021
San Diego County Office of Education, Innovation and ITS (datascience@innovatesd.net)
Application development: Leapfrog Technology, Inc. (DBA Eduphoric)

Contents

1. Introduction

This SDCOE application that can be deployed to a Linux or Windows environment. This document will help you understand the steps involved in deploying the SDCOE application in the Windows environment using IIS server.

Machine specification
RAM: 2GB
Hard Drive: 60GB
Operating System: Windows 10 (local playground) or Windows Server
Other Dependencies:

Git bash
node v14.15.1
npm 6.14.8
pm2 4.5.0
IIS Server
MS SQL

Deployment Notes

Note: There are sections in this document where a URL, domain name, user name, password, IP address or email address are highlighted in light orange. This signifies where each person or organization installing this application will substitute their own information.

Note to consider before beginning an installation using this documentation:

- These steps are described as installed on a Windows 10 Home environment. Few steps regarding setting up permissions and installation may differ in the case of Windows Server environment.
- There are videos referenced in the documentation. These are available as of spring 2021 but may be moved over time and might be inaccessible.

2. Project Setup and Dependencies installation

This section assumes that you have a GitHub account already created, and that you have installed Git for Windows. Git Bash comes included as part of the Git For Windows package. Download and install Git For Windows like other Windows applications. Once downloaded find the included .exe file and open to execute Git Bash.

a. Adding a new SSH key to your GitHub account

Follow the instructions in the documentation given below.
Adding SSH Key

b. Clone the repository and move the project in /sdcoe-transcript directory

Open git-bash in Desktop and run the following command to clone the project in Desktop.

```
$ git clone git@github.com:leapfrogtechnology/sdcoe-transcript.git
```

This will create the **sdcoe-transcript** folder in the Desktop. This contains all the modules/folders for the SDCOE project.

The front-facing packages of the sdcoe-transcript project are the transcript-api, the verification-api and the web-client packages. Other utilities exist in this git repository to facilitate the backend work of these front-facing packages.

Description of each of the packages in brief:

Design
- This folder contains the main design elements that have been used for the SDCOE front end.

Docs
- This folder is the main documentation module
- Contains Markdown files

Email Utils
- This folder contains libraries and templates for sending out SDCOE emails
- Is a node package with handlebar templates

PDF Utils
- This folder contains libraries to modify a PDF and its metadata
- Is a node package for with code mainly used for handling pdf metadata

Scripts
- This folder contains scripts for deployment, creating services and for running background schedules bash scripts for deployment; cron files for scheduling

Security

- This folder contains utilities for transcript security such as key generation and blockchain
- Node package with libraries for DID management, key generation, blockchain interaction, JWT Signing

Transcript API
- This is the express api module for requesting transcripts
- Model View Controller express app; Knex based migration scripts

Transcript Backend
- This is the module that deals with interaction with ODS and for creating standard JSON and pdf files
- Node package with JS and handlebar files

Verification API
- This is the express api module for verifying transcripts
- Node express related code

Web Client
- This is the React JS web application

a. Setup necessary dependencies

Install Node.JS, PM2, ganache-cli, yarn

Install Node.Js

Download Node.js Installer for Windows.

- Go to the site https://nodejs.org/en/download/ and download the necessary binary files.
- Run the installation
- Verify Installation by checking the version by running the following commands:

```
$ node -v
$ npm -v
```

Install PM2, ganache-cli, yarn from https://www.npmjs.com/

After installing Node, the following command to install to pm2, ganache-cli & yarn. Open a git bash and run the following command:

```
$ npm install pm2 -g
```

```
$ npm -g install ganache-cli
```

```
$ npm i -g yarn
```

Install all the packages globally. And you can verify the installation by checking its versions.

Note that you may need to setup system environment variables and ad a SYSTEM variable as

PM2_HOME

Pointing to the following or like location:

C:\Users\Administrator\AppData\Roaming\npm\node_modules\pm2

Per: "This is crucial for every further step. The key point for PM2 on Windows is to set up Windows an environment variable PM2_HOME at system level. If you miss that, you'll get into trouble." Reference:

https://blog.cloudboost.io/nodejs-pm2-startup-on-windows-db0906328d75

To check/verify pm2 version:

$ pm2 --version

```
[PM2] Spawning PM2 daemon with
pm2_home=C:\Users\Administrator\AppData\Roaming\npm\node_modules\pm2
[PM2] PM2 Successfully daemonized
4.5.6
```

To install yarn

$ yarn

```
yarn install v1.22.10
info No lockfile found.
[1/4] Resolving packages…
[2/4] Fetching packages…
[3/4] Linking dependencies…
[4/4] Building fresh packages…
success Saved lockfile.
Done in 0.08s.
```

Video Link: 01 Project Setup and Installing Dependencies

3. Security Setup

a. Run ganache-cli using PM2

There is a script to run the ganache server in the cli folder inside the security module, navigate to the location on your system, for instance:

```
cd /c/desktop/sdcoe-transcript/security
```

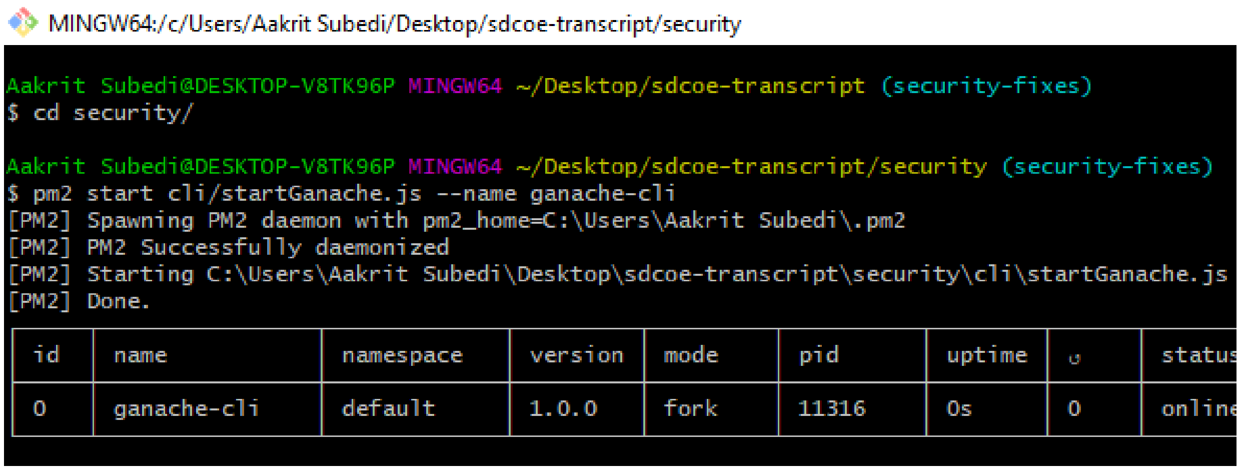
This script will internally runs the ganache-cli command, and start the ganache-cli at the port specified in the .env file of the security module

Populate the values for GANACHE_PORT, GANACHE_MNEMONIC and GANACHE_ACCOUNTS in the .env file. Also add the path and file name for the BLOCKCHAIN_PATH and BLOCKCHAIN_KEY_FILE.

```
...
# A path where secret files are stored
SECURED_FOLDER=secured-folder
..
# Blockchain keys file paths
BLOCKCHAIN_KEY_FILE=keys.json
BLOCKCHAIN_PATH=blockchain-info
...
# Ganache Info
GANACHE_PORT=8545
GANACHE_MNEMONIC=sdcoe-project
GANACHE_ACCOUNTS=1
```

Now, after setting up the values for those variables, run the following command to start ganache-cli.

```
$ cd security

# install dependencies
$ yarn

$ pm2 start cli/startGanche.js --name ganache-server
```



MINGW64:/c/Users/Aakrit Subedi/Desktop/sdcoe-transcript/security

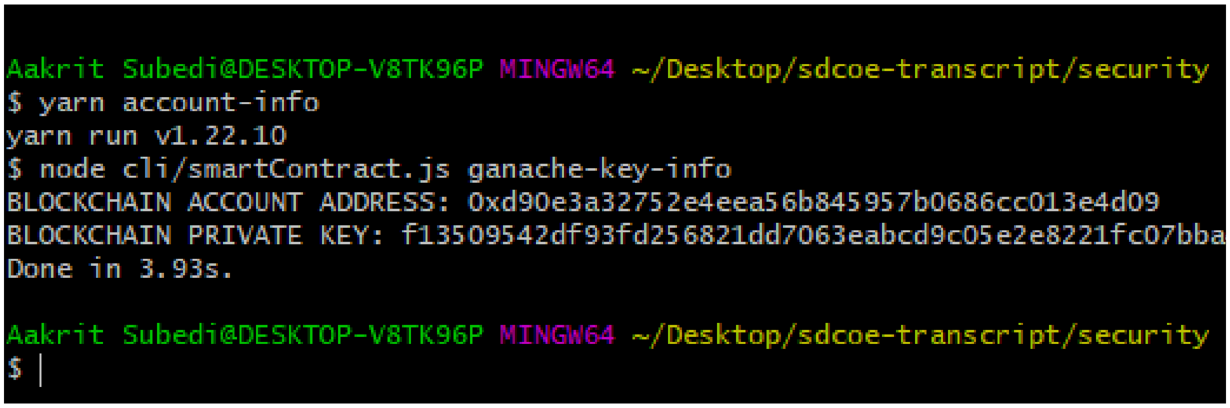
```
Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript (security-fixes)
$ cd security/

Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript/security (security-fixes)
$ pm2 start cli/startGanache.js --name ganache-cli
[PM2] Spawning PM2 daemon with pm2_home=C:\Users\Aakrit Subedi\.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting C:\Users\Aakrit Subedi\Desktop\sdcoe-transcript\security\cli\startGanache.js
[PM2] Done.
```

id	name	namespace	version	mode	pid	uptime	↺	status
0	ganache-cli	default	1.0.0	fork	11316	0s	0	online

The above command will start the ganache server at the given port. All the account info will be stored in the blockchain directory in file named **keys.json**

You can see the account info in keys.json file or simply by running the following command:

```
$ yarn account-info
```



```
Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript/security
$ yarn account-info
yarn run v1.22.10
$ node cli/smartContract.js ganache-key-info
BLOCKCHAIN ACCOUNT ADDRESS: 0xd90e3a32752e4eea56b845957b0686cc013e4d09
BLOCKCHAIN PRIVATE KEY: f13509542df93fd256821dd7063eabcd9c05e2e8221fc07bba
Done in 3.93s.

Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript/security
$ |
```

Update the ETHEREUM_ACCOUNT_ADDRESS & ETHEREUM_PRIVATE_KEY with the value of account address and private key received in the step above. Also add the ETHEREUM_NODE_URL to http://localhost:8545 in the .env file.

```
# Ethereum endpoint and the did registry address
ETHEREUM_NODE_URL=http://localhost:8545

# Ethereum account and its private key used
ETHEREUM_ACCOUNT_ADDRESS=0xd90e3a32752e4eea56b845957b0686cc013e4d09
ETHEREUM_PRIVATE_KEY=f13509542df93fd256821dd7063eabcd9c05e2e8221fc07bba4cb304a91d
d69d
```

Example of the .env file:

```
# Ethereum endpoint and the did registry address
ETHEREUM_NODE_URL=${BLOCKCHAIN_NODE_URL}
DID_REGISTRY_ADDRESS=${BLOCKCHAIN_DID_REGISTRY_ADDRESS}

# Ethereum account and its private key used
ETHEREUM_ACCOUNT_ADDRESS=${BLOCKCHAIN_ACCOUNT_ADDRESS}
ETHEREUM_PRIVATE_KEY=${BLOCKCHAIN_PRIVATE_KEY}

# A path where secret files are stored
SECURED_FOLDER=secured-folder

# Blockchain keys file path
BLOCKCHAIN_KEY_FILE=keys.json
BLOCKCHAIN_PATH=blockchain-info

# Blockchain delegate key
```

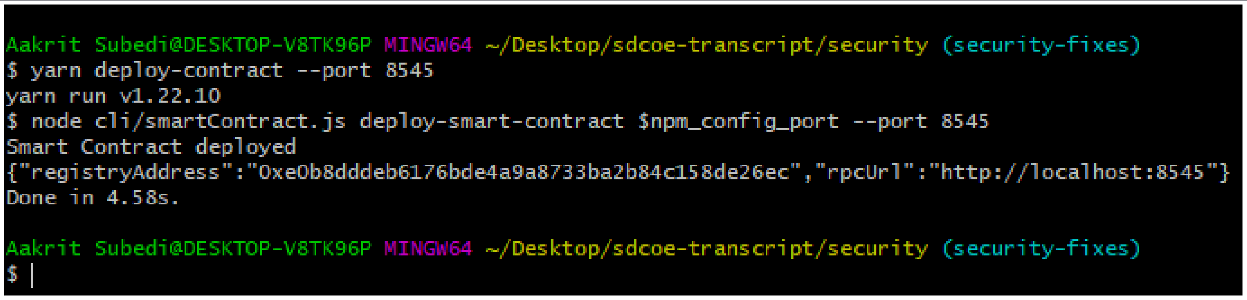
```
KEY_ID=${BLOCKCHAIN_DID_DELEGATE_KEY}

# Ganache Info (note one of the MNENOMICS is misspelled, so repeated both
spellings)
GANACHE_PORT=8545
GANACHE_MNENOMIC=sdcoe-project
GANACHE_MNEMONIC=sdcoe-project
GANACHE_ACCOUNTS=1
```

b. Deploy the smart contract

Now, the next step is to deploy the smart contract to the local ethereum network.

Run the following command to deploy the smart contract to the network.

```
$ yarn deploy-contract --port 8545
```



```
Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript/security (security-fixes)
$ yarn deploy-contract --port 8545
yarn run v1.22.10
$ node cli/smartContract.js deploy-smart-contract $npm_config_port --port 8545
Smart Contract deployed
{"registryAddress":"0xe0b8dddeb6176bde4a9a8733ba2b84c158de26ec","rpcUrl":"http://localhost:8545"}
Done in 4.58s.

Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript/security (security-fixes)
$ |
```

This will give you the BLOCKCHAIN_DID_REGISTRY_ADDRESS, copy that info and paste it in the .env file.

```
# Ethereum endpoint and the did registry address
ETHEREUM_NODE_URL=http://localhost:8545
DID_REGISTRY_ADDRESS=0xe0b8dddeb6176bde4a9a8733ba2b84c158de26ec
...
```

c. Signer

- **Generating new RSA key value pair**
  We had used openssl to generate the RSA private and public key. Instead of using a similar application, we will use a script to generate the private and public key.

```
$ yarn new-rsa --label sdcoe
```

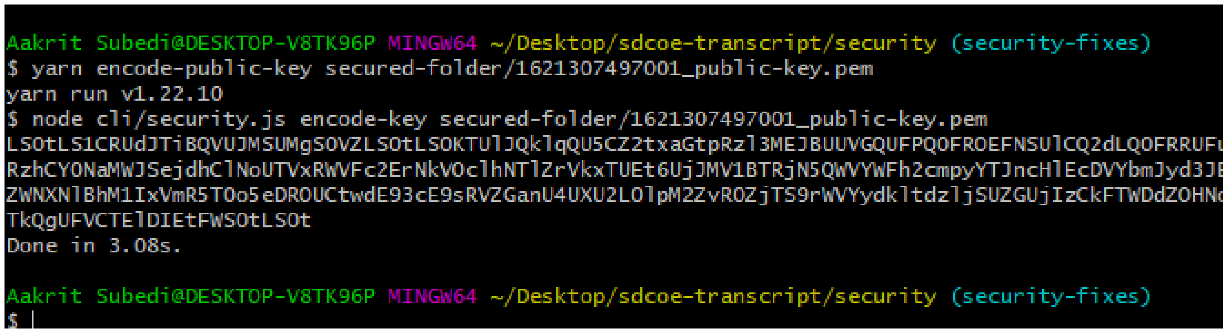
Note you may need to use this command if above does not work:

```
yarn new-rsa sdcoe
```

This will create the new key value pair in the secured directory as configured in the .env file, which you can move to the separate folder and rename if necessary. The final path for the private key should be specified in the .env file of the transcript-api module.

- **Base64 encode public key**
  Now, encode the public key using following command:

```
$ yarn encode-public-key secured-folder/<public-key-filename>

# EXAMPLE
$ yarn encode-public-key  secured-folder/public-key.pem
```



```
Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript/security (security-fixes)
$ yarn encode-public-key secured-folder/1621307497001_public-key.pem
yarn run v1.22.10
$ node cli/security.js encode-key secured-folder/1621307497001_public-key.pem
LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlJQklqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FROEFNSUlCQ2dLQ0FRRUF
RzhCYONaMWJSejdhClNoUTVxRWVFc2ErNkVOclhNTlZrVkxTUEt6UjJMV1BTRjN5QWVYWFh2cmpyYTJncHlEcDVYbmJyd3J
ZWNXNlBhM1IxVmR5TOo5eDROUCtwdE93cE9sRVZGanU4UXU2L0lpM2ZvR0ZjTS9rWVYydkltdzljSUZGUjIzCkFTWDdZOHN
TkQgUFVCTElDIEtFWS0tLS0t
Done in 3.08s.

Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript/security (security-fixes)
$ |
```

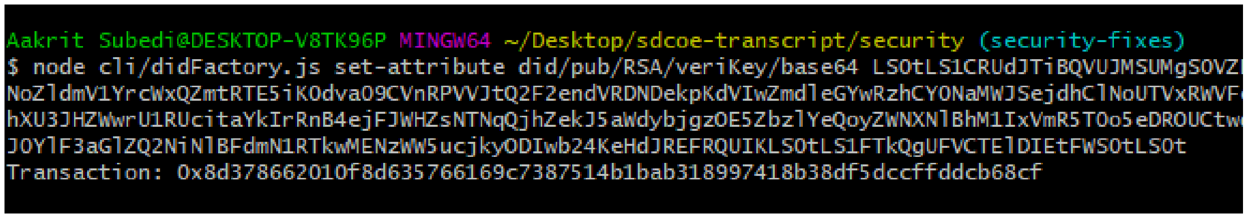
This will return the base64 encoded public key.

- **Set Attribute**
  Now, we need to add the decoded public key to the blockchain account.

Run the following command to set-attribute.

```
$ node ./cli/didFactory.js set-attribute did/pub/RSA/veriKey/base64
"$SDCOE_VERIFIER_PUBLIC_KEY_BASE_64"
```

where, **SDCOE_VERIFIER_PUBLIC_KEY_BASE_64** refers to the base64 encoded version of the decoded public key from the above step.



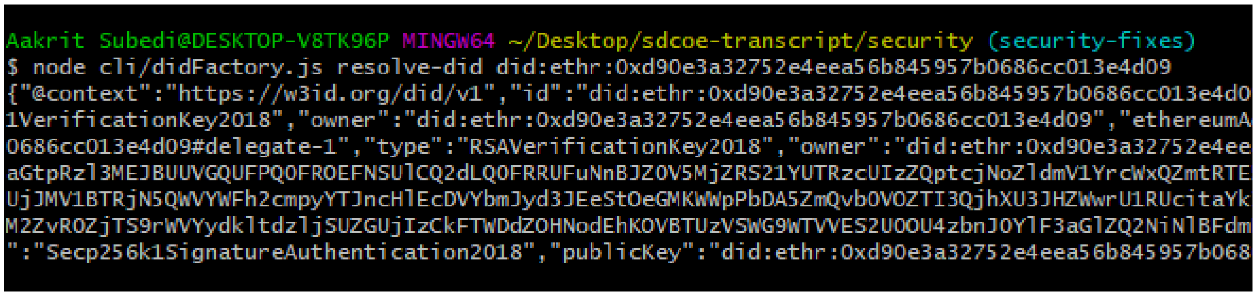
```
Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript/security (security-fixes)
$ node cli/didFactory.js set-attribute did/pub/RSA/veriKey/base64 LS0tLS1CRUdJTiBQVUJMSUMgS0VZ
NoZldmV1YrcWxQZmtRTE5iK0dva09CVnRPVVJtQ2F2endVRDNDekpKdVIwZmdleGYwRzhCYONaMWJSejdhClNoUTVxRWVF
hXU3JHZWwrU1RUcitaYkIrRnB4ejFJWHZsNTNqQjhZekJ5aWdybjgzOE5ZbzlYeQoyZWNXNlBhM1IxVmR5TOo5eDROUCtw
JOYlF3aGlZQ2NiNlBFdmN1RTkwMENzWW5ucjkyODIwb24KeHdJREFRQUIKLS0tLS1FTkQgUFVCTElDIEtFWS0tLS0t
Transaction: 0x8d378662010f8d635766169c7387514b1bab318997418b38df5dccffddcb68cf
```

- **Resolve the DID to get KEY_ID**
  Run the following command to resolve DID

```
$ node ./cli/didFactory.js resolve-did
"did:ethr:$BLOCKCHAIN_ACCOUNT_ADDRESS"
```

BLOCKCHAIN_ACCOUNT_ADDRESS refers to the address that we received after deploying the contract.

At this step we have configured the .env file of the security module.



```
Aakrit Subedi@DESKTOP-V8TK96P MINGW64 ~/Desktop/sdcoe-transcript/security (security-fixes)
$ node cli/didFactory.js resolve-did did:ethr:0xd90e3a32752e4eea56b845957b0686cc013e4d09
{"@context":"https://w3id.org/did/v1","id":"did:ethr:0xd90e3a32752e4eea56b845957b0686cc013e4d0
1VerificationKey2018","owner":"did:ethr:0xd90e3a32752e4eea56b845957b0686cc013e4d09","ethereumA
0686cc013e4d09#delegate-1","type":"RSAVerificationKey2018","owner":"did:ethr:0xd90e3a32752e4ee
aGtpRzl3MEJBUUVGQUFPQ0FROEFNSUlCQ2dLQ0FRRUFuNnBJZ0V5MjZRS21YUTRzcUIzZQptcjNoZldmV1YrcWxQZmtRTE
UjJMV1BTRjN5QWVYWFh2cmpyYTJncHlEcDVYbmJyd3JEeStOeGMKWWpPbDA5ZmQvb0VOZTI3QjhXU3JHZWwrU1RUcitaYk
M2ZvR0ZjTS9rWVYydkltdzljSUZGUjIzCkFTWDdZOHNodEhKOVBTUzVSWG9WTVVES2U0OU4zbnJ0OYlF3aGlZQ2NiNlBFdm
":"Secp256k1SignatureAuthentication2018","publicKey":"did:ethr:0xd90e3a32752e4eea56b845957b068
```

Copy the **id** <did:ethr:....#delegate-1> and paste it in **key_id** of the .env of the security module.

```
...
# Blockchain delegate key
KEY_ID=did:ethr:0xd90e3a32752e4eea56b845957b0686cc013e4d09#delegate-1
...
```

The pattern for the KEY_ID is "did:ethr:$BLOCKCHAIN_ACCOUNT_ADDRESS#delegate-1".

Video Link: [02 Setting Up Security Utils]

4. Database Configuration

a. Configure the MS SQL database

- Go to URL: https://www.microsoft.com/en-in/sql-server/sql-server-downloads
  And download the **Developer edition** for installation.
- Install the MS SQL server selecting Basic option as it has all default configuration required to run MS SQL.
- Select the desired location for installation
- Once installation is completed successfully, the below screen will appear.

b. Create a Login

To create a login, Navigate to Security > Logins
In the next screen, Enter

1. Login Name
2. Select SQL Server authentication
3. Enter Password
4. Click Ok

You can also create a login using the T-SQL command.

```
$ CREATE LOGIN MyLogin WITH PASSWORD = '123';
```

c. Create a User

A user is an account that you can use to access the SQL server. To create users, you can use any of the following two ways:

d. Using SQL Server Management Studio

1. Connect to SQL Server then expand the Databases folder from the Object Explorer.
2. Identify the database for which you need to create the user and expand it.
3. Expand its Security folder.
4. Right-click the Users folder then choose "New User…"

You can create a new USER using the T-SQL's create user command. The command takes the following syntax:

```
$ create user <user-name> for login <login-name>
```

e. Assigning Permission to a User

Connect to your SQL Server instance and expand the folders from the Object Explorer as shown below. Right click on the name of the user, that is, sdcoe then choose Properties.

In the next screen,

- Click the Securables option from the left.
- Click on Search

In the next window,

- Select "All Objects belonging to the Schema."

- Select Schema name as "dbo”
- Click OK

Next Step,

- Identify Table you want to Grant Permission
- In Explicit Permission select Grant
- Click Okay

The permission will be granted!

Ref: https://www.guru99.com/sql-server-create-user.html

Also, allow the database to be accessed by other login methods.

f. Create application database

Create a database for the application: sdcoetranscript.

g. Add configuration info in the .env file of the transcript-api

Populate the value to database config in the transcript-api module, which is the folder in the sdcoe-transcript project directory. Transcript API module is the express api for requesting a transcript. It consists of the Model, View & Controller which communicates the DATABASE just configured in the above step. In order to facilitate the communication between the transcript-api module and the MS SQL database we need to configure it in the transcript-api env.

The .env file for the transcript-api with the database configuration should look like:

```
…
# DATABASE Config
DB_PORT=1434
DB_HOST=localhost
DB_USER=my_username
DB_PASSWORD=my_password
DB_NAME=sdcoe
DB_CLIENT=mssql
...
```

5. Transcript-API

This module contains the code for REST API for transcript requesting.
Read More

As, we have already added the database configuration in the .env file of the transcript-api. We can now migrate the database schema and seed the pre configured data to the database.

a. Installing Dependencies

Navigate to the transcript-api module within the project, which is a folder containing all business logic, model, database table schema for requesting transcript.

```
$ cd transcript-api
```

Then, run the following command to install the dependencies for the module. This will install all the dependencies in the node_modules folder.

```
# installing dependencies
$ yarn
```

Note: In one installation instance, needed to run the yarn install twice. The second time worked better.

As, we have already configured the database config in the .env file of the transcript-api. The database config would look like the following config.

```
…
# DATABASE Config
DB_PORT=1434
DB_HOST=localhost
DB_USER=my_username
DB_PASSWORD=my_password
DB_NAME=sdcoe
DB_CLIENT=mssql
...
```

b. Migrate and Seed

Migrations are a type of version control for your database. Migrations are typically paired with the Schema Builder to easily manage your application's database table schema.

The migration will be placed in your transcript-api/src/migrations folder, and will contain a timestamp which allows the framework to determine the order of the migrations.

Creating new Migrations
These are the commands to create a new migration file.

```
$ yarn make:migration <name>
```

Running Migrations
Migrate the existing table schema

```
$ yarn migrate
```

Knex includes the ability to seed your database with test data using seed classes. All seed classes are stored in the transcript-api/src/seeds directory.

Creating new Seeds
These are the commands to create a new seed file.

```
$ yarn make:seeder <name>
```

Running Seeds
Seeds the existing table data schema:

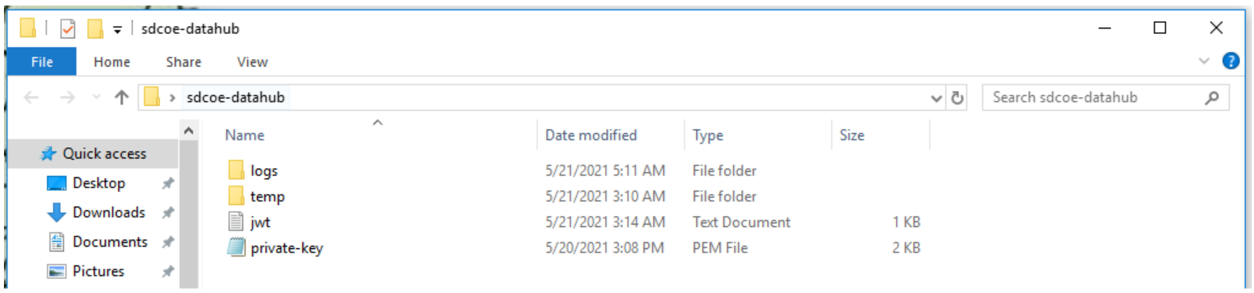
```
$ yarn seed
```

c. Configure the remaining .env

Create two folders, one for the temp files and other for maintaining logs.

For an installation used in demonstration and testing, both of the folders above are placed in the sdcoe-datahub folder created while configuring the security module in the above step.

Also, create a text file using an editor called add the jwt.txt file containing the jwt secret in the same folder. The jwt secret is a key created at this time, and for testing, “thisismyjwtkey” is an example.



Name	Date modified	Type	Size
logs	5/21/2021 5:11 AM	File folder	
temp	5/21/2021 3:10 AM	File folder	
jwt	5/21/2021 3:14 AM	Text Document	1 KB
private-key	5/20/2021 3:08 PM	PEM File	2 KB

then , add path to those directories and file in the .env

The final .env for transcript-api in my case looks like the following sample. Refer to the following sample to configure it.

```
# Log and temp PDF directory
LOG_DIR=C:\Users\Username\Desktop\sdcoe-datahub\logs
TMP_PDF_LOCATION=C:\Users\Username\\Desktop\sdcoe-datahub\temp

# APPLICATION
APP_HOST=127.0.0.1
APP_PORT=3000
APP_BASE_URL=http://127.0.0.1:3000/api

# DATABASE Config
DB_PORT=1434
DB_HOST=localhost
DB_USER=mysdcoe
DB_PASSWORD=admin
DB_NAME=sdcoe
DB_CLIENT=mssql

# SMTP Mail Config
SMTP_HOST=
SMTP_PORT=
SMTP_AUTH_USER=
SMTP_AUTH_PASSWORD=
SMTP_FROM=

# SDCOE static logo path and contact me mail
SDOCOE_LOGO_URI=https://yourdomain/logo250px-224x173.png
MAIL_TO=communication@yourdomain.net

# Security
DID_KEY=did:ethr:0xd90e3a32752e4eea56b845957b0686cc013e4d09#delegate-1

# RSA: Path to file containing the signer private key
RSA_PRIVATE_KEY_FILE=C:\Users\Username\Desktop\sdcoe-datahub\private-key.pem

# JWT: Path to file containing the secret key
JWT_SECRET_KEY_FILE=C:\Users\Username\Desktop\sdcoe-datahub\jwt.txt

# JWKS_KEYS to validate the token from microsoft
JWKS_URI=https://login.microsoftonline.com/common/discovery/keys?appid=79297b4d-a602-4206-9b
53-9ceb9633e6e0

# IMAGE_URL: Url of the images that transcript & certificate uses
ROP_STICKER=http://localhost:3000/api/assets/rop-sticker.png
ROP_COLLEGE_ASSOCIATE_STICKER =
http://localhost:3000/api/assets/college-assoicate-sticker.png
SUPERITENDENT_SIGNATURE=http://localhost:3000/api/assets/signature-director.png
SENIOR_DIRECTOR_FIRST=http://localhost:3000/api/assets/senior-director-one.png
```

```
SENIOR_DIRECTOR_SECOND=http://localhost:3000/api/assets/senior-director-signature-two.png
```

Note: These env variables are used for image URL that needs to be hosted somewhere.
ROP_STICKER = link of the image/sticker for ROP which is present at the bottom-left on the ROP certificate.

ROP_COLLEGE_ASSOCIATE_STICKER = Present at the bottom-right on the ROP Certificate.

SUPERINTENDENT/DIRECTOR signatures at the bottom-center on the ROP certificate.

d. Start the transcript-api server using PM2

At this point we have already configured the database and migrated the table schema. Then after migration, we also seeded the preconfigured value to the tables in the sdcoe database and configured the .env file referring to the sample above.

Now, we can run the following command to build the production files and run it using PM2. Run all the commands from the transcript-api (folder - if needed, use a command like this to change to the folder: $ cd /c/desktop/sdcoe-transcript/transcript-api

```
# installing dependencies
$ yarn

# making production build
$ yarn build
```

This will create the dist folder containing all modules and dependencies. Now run the index.js file in the dist folder to start the server.

```
$ pm2 start dist/index.js --name transcript-api
```

If this works correctly, one should be able to point a browser to the URL and port specified in the .env file, and adding the api and a specific api request to the end, see results:

127.0.0.1:3000/api/districts

Video Link: 03 Transcript API

6. Multiple ODS config

Previously, the ods config was also set in the .env file of the transcript-api module. Now, in order to support the multiple-ods feature, we have updated the existing table **edfi_ods** in the sdcoe database with new columns. Now, we can simply run the INSERT query in the database console or any database management tool.

Insert Query

```
INSERT INTO edfi_ods(client_id, client_secret, base_url) VALUES('clientId', 'clientSecret', 'baseURL');
```

Example Query

```
INSERT INTO edfi_ods(client_id, client_secret, base_url) VALUES('KEY/ID', 'SECRET', 'https://youredifiapi.com');
```

Video Link: 04 Multiple ODS

7. Verification-API

This is the express api module for verifying transcripts. This module contains the web api that allows verification of a PDF transcript uploaded by the user. For the verification, this module uses the pdf-utils library.
Read More

a. Installing dependencies

Navigate to the transcript-api module, which is a folder containing all business logic, model, database table schema for requesting transcript.

```
$ cd verification-api
```

Then, run the following command to install the dependencies for the module. This will install all the dependencies in the node_modules folder.

```
# installing dependencies
$ yarn
```

b. Configure .env file

Add a folder named temp_pdf to store temp uploaded pdf files for verification . For this demo installation, I have added the folder in the sdcoe-datahub folder created while configuring the security module in the above step. Then , add a path to those directories and file in the .env file.

The final .env for verification-api in my case looks like the following sample. Refer to the following sample to configure it.

```
# Application
APP_NAME=SDCOE Verification API
APP_PORT=5000
APP_HOST=127.0.0.1
# Log
LOG_LEVEL=C:\Users\Username\Desktop\sdcoe-datahub\logs
LOG_DIR=C:\Users\Username\Desktop\sdcoe-datahub\logs

# Multer upload
MULTER_UPLOAD_DIRECTORY=C:\Users\Username\Desktop\sdcoe-datahub\temp_pdf

# Blockchain related values
ETHEREUM_NODE_URL=http://localhost:8545
DID_ADDRESS=0xe0b8dddeb6176bde4a9a8733ba2b84c158de26ec
DID_KEY=did:ethr:0xd90e3a32752e4eea56b845957b0686cc013e4d09#delegate-1
```

c. Start the verification-server using PM2

Now, we can run the following command to build the production files and run it using PM2. Run all the commands from the verification-api.

```
# installing dependencies
$ yarn

# making production build
$ yarn build
```

This will create the dist folder containing all modules and dependencies. Now run the index.js file in the dist folder to start the server.

```
$ pm2 start dist/index.js --name verification-api
```

Video Link: 05 Verification API

8. Web Client

This module includes code for frontend applications built with React.
Read More

a. Installing dependencies

Navigate to the transcript-api module, which is a folder containing all business logic, model, database table schema for requesting transcript.

```
$ cd web-client
```

Then, run the following command to install the dependencies for the module. This will install all the dependencies in the node_modules folder.

```
# installing dependencies
$ yarn
```

b. Configure .env file

The final .env for web-client in my case looks like the following sample. Refer to the following sample to configure it. Note the highlighted sections which should be replaced with details from your specific installation.

```
# Basic Frontend Config
REACT_APP_BASE_NAME=SDCOE transcript
REACT_APP_ENV=production

# API and Verify API endpoint link
REACT_APP_API_BASE_URI=http://localhost:3000/api
REACT_APP_VERIFICATION_API_URI=http://localhost:5000/api

# Microsoft Redirect URI and logout redirect URI
REACT_APP_LOGIN_URI=https://login.microsoftonline.com/common/oauth2/v2.0/authorize?client_id=
79297b4d-a602-4206-9b53-9ceb9633e6e0&response_type=token+id_token&redirect_uri=https://subdo
main.localhost:9080/auth/callback&scope=user.read+openid+profile+email&response_mode=fragment
&state=12345&nonce=678910
```

```
REACT_APP_REDIRECT_URI=http://subdomain.localhost:9080
```

Login URL:
https://login.microsoftonline.com/common/oauth2/v2.0/authorize?client_id=79297b4d-a602-4206-9b53-9ceb9633e6e0&response_type=token+id_token&redirect_uri=https://subdomain.localhost:9080/auth/callback&scope=user.read+openid+profile+email&response_mode=fragment&state=12345&nonce=678910

Note: You need to add the login url in Microsoft console.

c. Build the project

Finally build the react application.

```
# installing dependencies
$ yarn build
```

This will create all the necessary assets, dependencies for the web-client in the build folder. Now, we just need to map to this folder in IIS.

Video Link: 07 Web Client

9. Configure IIS and setup reverse proxy

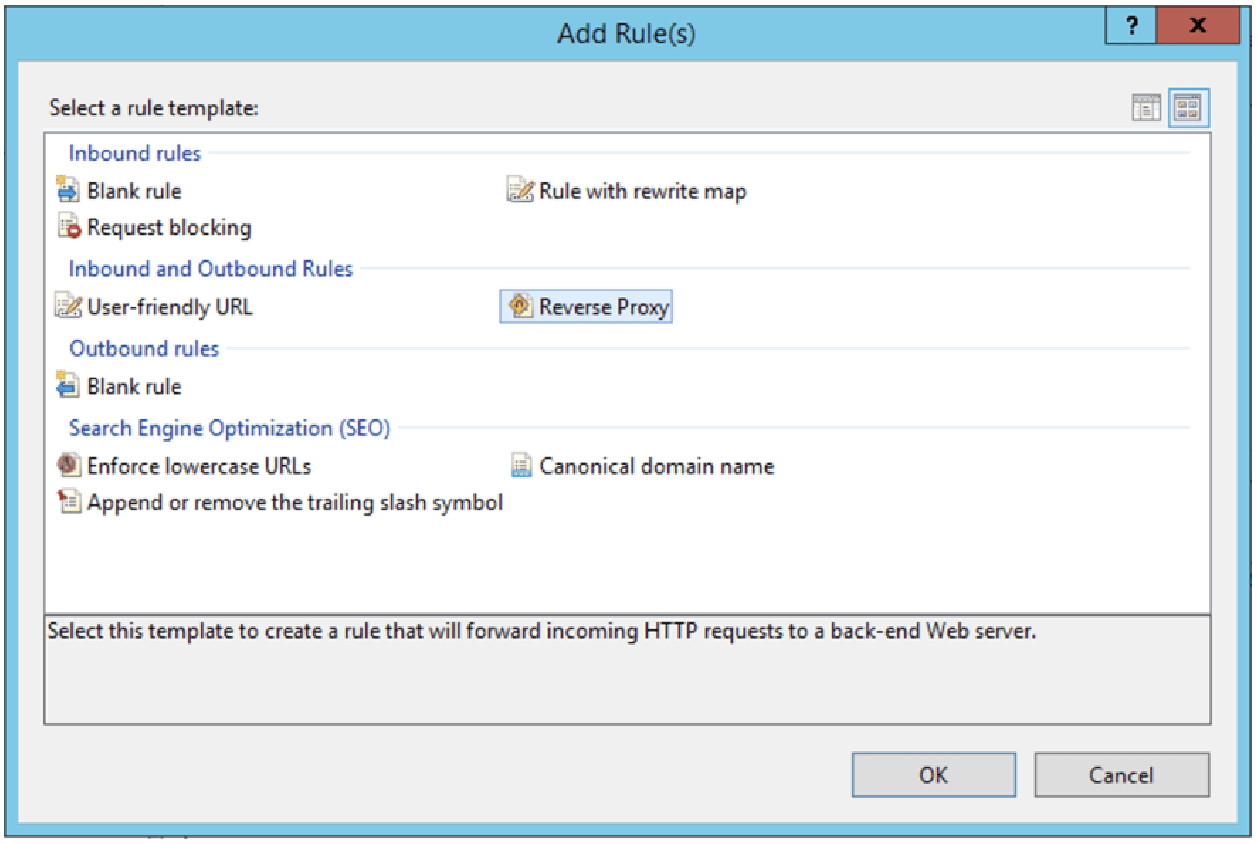
IIS Server must be installed and active on the server.

Add sites for transcript-api and verification-api in IIS server and add reverse-proxy to map the respective ports specified in the above step.

Install the URL Rewrite extension and Application Request Routing(ARR) which allows you to define rules to enable URLs that are easier for users to remember and for search engines to find. Once these extensions are installed, you can begin configuring IIS.

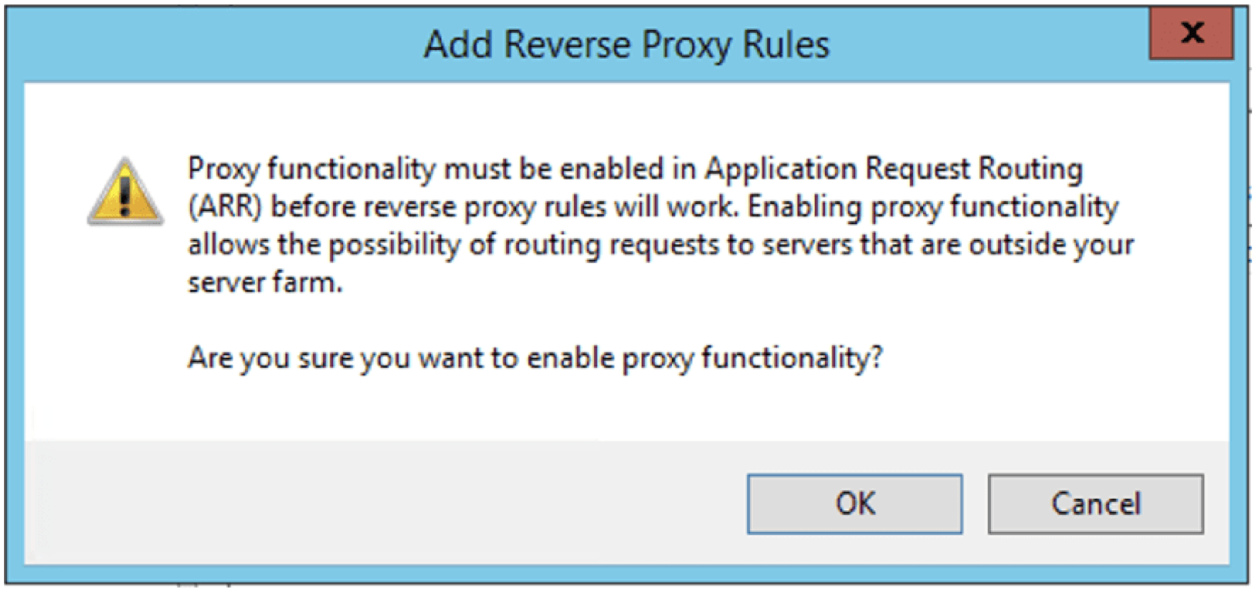
1. Open the Internet Information Services (IIS) Manager by opening the run window and typing the inetmgr command. Select the site for which you want to set up the reverse proxy and open the URL Rewrite extension.

2. Add a new rule and select the Reverse Proxy template.



Select this template to create a rule that will forward incoming HTTP requests to a back-end Web server.

3. Enable proxy functionality when you are prompted for it.



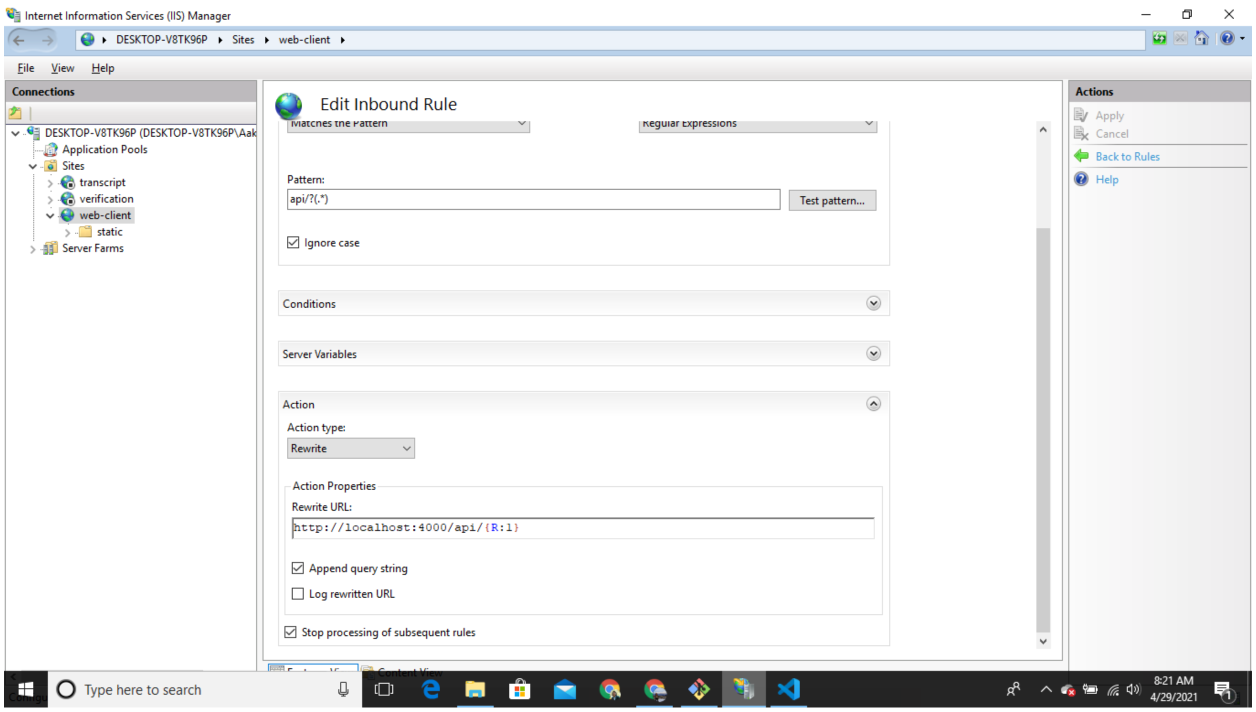
Proxy functionality must be enabled in Application Request Routing (ARR) before reverse proxy rules will work. Enabling proxy functionality allows the possibility of routing requests to servers that are outside your server farm.

Are you sure you want to enable proxy functionality?

4. Add the address of your node.js website, don't forget to include the port, to the reverse proxy rules and also add the url pattern

- `api/?(.*)` for transcript-api
- `api/verify/?(.*)` for verification-api

as shown in the screenshot below


Internet Information Services (IIS) Manager

DESKTOP-V8TK96P ▸ Sites ▸ web-client ▸

Connections: DESKTOP-V8TK96P (DESKTOP-V8TK96P\Aak — Application Pools — Sites — transcript — verification — web-client — static — Server Farms

Edit Inbound Rule

Matches the Pattern — Regular Expressions

Pattern: api/?(.*)

☑ Ignore case

Conditions

Server Variables

Action

Action type: Rewrite

Action Properties

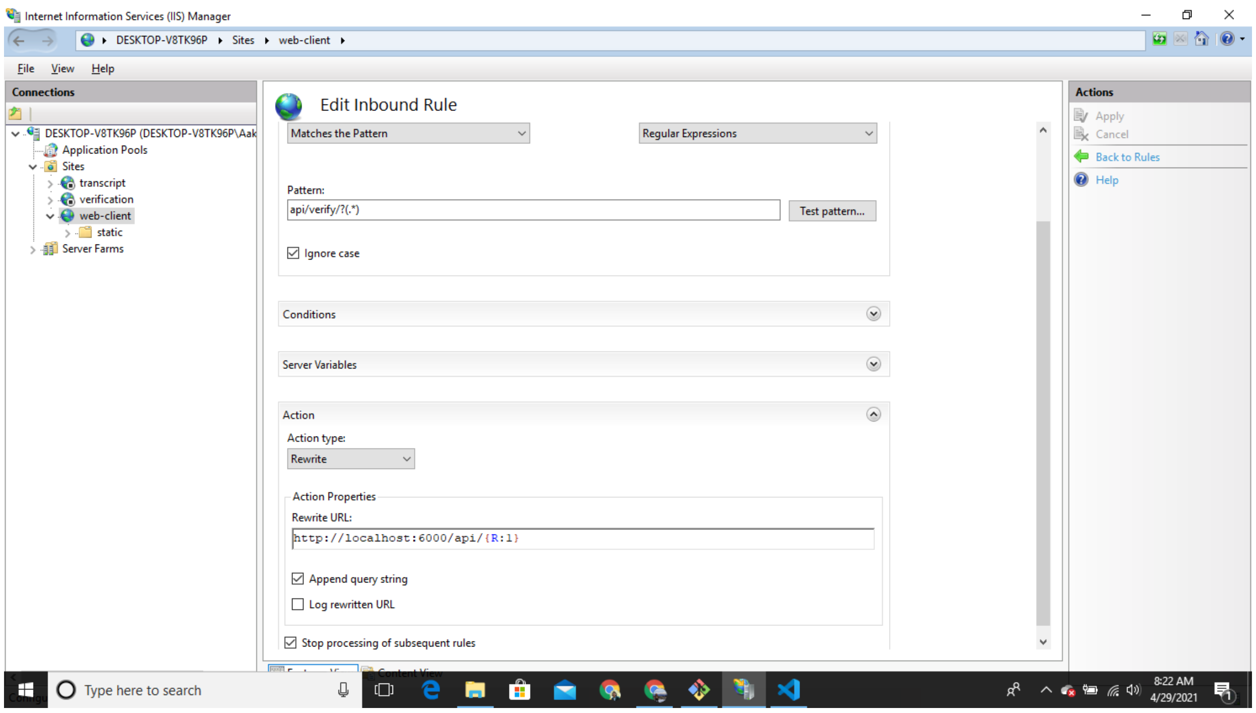
Rewrite URL: `http://localhost:4000/api/{R:1}`

☑ Append query string

☐ Log rewritten URL

☑ Stop processing of subsequent rules

Actions: Apply, Cancel, Back to Rules, Help


Internet Information Services (IIS) Manager

DESKTOP-V8TK96P ▸ Sites ▸ web-client ▸

Connections: DESKTOP-V8TK96P (DESKTOP-V8TK96P\Aak — Application Pools — Sites — transcript — verification — web-client — static — Server Farms

Edit Inbound Rule

Matches the Pattern — Regular Expressions

Pattern: api/verify/?(.*)

☑ Ignore case

Conditions

Server Variables

Action

Action type: Rewrite

Action Properties

Rewrite URL: `http://localhost:6000/api/{R:1}`

☑ Append query string

☐ Log rewritten URL

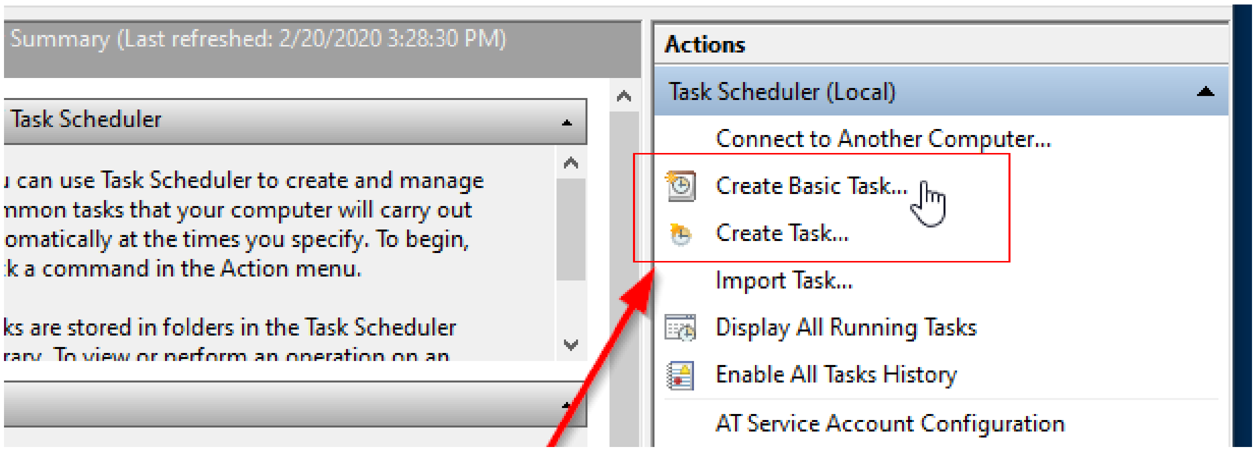
☑ Stop processing of subsequent rules

Actions: Apply, Cancel, Back to Rules, Help

Also, add permission to the respective sites folder.

10. Setup schedule

- Open Task Scheduler
  WIN+R, taskschd.msc
- Start the task creation process by clicking on "Create Basic Task" or "Create Task" in the sidebar


Summary (Last refreshed: 2/20/2020 3:28:30 PM)
Task Scheduler
Actions
Task Scheduler (Local)
Connect to Another Computer...
Create Basic Task...
Create Task...
Import Task...
Display All Running Tasks
Enable All Tasks History
AT Service Account Configuration

u can use Task Scheduler to create and manage
nmon tasks that your computer will carry out
omatically at the times you specify. To begin,
k a command in the Action menu.

ks are stored in folders in the Task Scheduler

- Then, Pick a trigger
- Add your Action
  - **"Program/script":**
    Here is where you plug in the path to the application you found by following step Find where the binary / application you need to run is stored.
    You can use **where npm** or **where yarn** from the command line to find the path

    Eg: **C:\Program Files (x86)\Yarn\bin\yarn.cmd**
  - **Add arguments**
    You should put whatever you would put after npm or yarn normally.
    In our case: **setup-cron**
    If you are concerned about keeping track of the results of what ran, you can also capture the result of anything spit out to the console by using **>> task_log.txt** or something like that.
  - **Use the start in (optional) field to specify the directory**
    Note: Please change the setting as per your requirement.

Ref:https://joshuatz.com/posts/2020/using-windows-task-scheduler-to-automate-nodejs-scripts/

11. Login Credentials

Each installation can use local authentication or Azure AD SSO authentication. There is detailed information about the differences within the "**Deployment Options Guide**."

Local authentication: If using local authentication users are configured in the USERS table within the application database.

Azure AD: If using Azure AD, set up an application and users within an Azure AD subscription and configure the application .env for these settings. Roles for each user, tied to usernames (emails) are configured in the users table of the database.

Roles within application:

- Staff User.
- District User.

12. Q&A

QUESTION: Do the steps involving the setup of Ganache (create ethereum account address, RSA keys, encoding keys) happen on the initial installation, or do these steps need repeated every time the system is restarted? or when an upgrade is changed?

Answer: Yes, you need to start the Ganache server again and you can use the same RSA keys. But you need to deploy the smart contract again and resolve it following the steps mentioned above in step no. 2.

But using the same mnemonics while starting the Ganache server the values for DID registry address, Ethereum Address and Private will be consistent.

```
ETHEREUM_NODE_PORT=8545
ETHEREUM_NODE_URL=HTTP://127.0.0.1:8545
DID_REGISTRY_ADDRESS=0xe0b8dddeb6176bde4a9a8733ba2b84c158de26ec

ETHEREUM_ACCOUNT_ADDRESS=0xd90E3a32752e4EeA56b845957B0686cC013e4D09
ETHEREUM_PRIVATE_KEY=0xf13509542df93fd256821dd7063eabcd9c05e2e8221fc07bba4cb304a9
1dd69d

SECURED_FOLDER=secured-folder

KEY_ID=did:ethr:0xd90E3a32752e4EeA56b845957B0686cC013e4D09#delegate-1
...
```

QUESTION: Do the .env files that currently exist in the different folders contain different settings, or are these all the same? If these are different, then each needs to be edited on initial installation to update the various settings, correct? Or, if these are all the same, doesn't the master .env file need to contain all settings needed throughout the application?

Answer:

Here is a list of all folders that contains .env files:

1. sdcoe-transcript/security
2. sdcoe-transcript/transcript-api
3. sdcoe-transcript/verification-api
4. sdcoe-transcript/web-client
5. sdcoe-transcript/pdf-utils
6. sdcoe-transcript/email-utils

Some of these do not need to be configured depending on the use of that module.

```
$ tree -a -P \.env* -L 2 --prune
.
├── email-utils
│   ├── .env
│   └── .env.example
├── pdf-utils
│   └── .env.example
├── security
│   ├── .env
│   └── .env.example
├── transcript-api
│   └── .env.example
├── transcript-backend
│   ├── .env
│   └── .env.example
├── verification-api
│   └── .env.example
└── web-client
    └── .env.example
```

Each of the folders seen above that have an .env.example can be used separately as a module. So we need to configure every module that we want to use. For the deployment process we only need to configure the ones highlighted above and listed below:

- security

- transcript-api
- verification-api
- web-client

and we do not need to configure

- email-utils
- pdf-utils
- transcript-backend

The configuration keys required for each module can be determined by looking at the content of the example file

```
$ cat email-utils/.env.example
# SMTP Email Config
SMTP_HOST=${SDCOE_SMTP_HOST}
SMTP_PORT=${SDCOE_SMTP_PORT}
SMTP_AUTH_USER=${SDCOE_SMTP_AUTH_USER}
SMTP_AUTH_PASSWORD=${SDCOE_SMTP_AUTH_PASSWORD}

# Sender Name and Email in format Sender Info <email>
SMTP_FROM=${SDCOE_SMTP_FROM}

# SDCOE static logo path and contact me mail
SDOCOE_LOGO_URI=${SDCOE_LOGO_URI}
MAIL_TO=${SDCOE_EMAIL}
```

For example the content of the example env contains what you see above. We need to make a copy of each .env.example by running the following command then **manually changing the content** of the .env file

```
$ cp .env.example .env

# Manually edit .env next
```

QUESTION: In web-client, in the .env file, there is a REACT_APP_ENV config, and currently the recommended value is REACT_APP_ENV=production
Is there any other value other than "production" and if so, what are the differences for the application?

The REACT_APP_ENV is generally used in the working mode of the application. We can switch between those values and reflect the different behaviour or configuration for the development and production. There are no hard and fast values for the REACT_APP_ENV in the sdcoe-transcript project. The possible values may be **development** and **production** to switch between the api endpoint url in the web-client.(may be other configurations are also controlled by the ENV mode)

It is a built-in support provided by create-react-app for configuration files for pre-defined environments i.e *development*, *test and production*. When we develop our application we signify the environment to development and production while deployment or build. We can set up different configs depending on environments so that it would use environment specific values.

QUESTION: In the .env file for web-client, the url for the web client in the Microsoft redirect for client_id 79297b4d-a602-4206-9b53-9ceb9633e6e0 is https://subdomain.localhost:9080 (see below). Is this a valid redirect_url registered with Microsoft, or how are other subdomains / URLs defined?

```
# Microsoft Redirect URI and logout redirect URI
REACT_APP_LOGIN_URI=https://login.microsoftonline.com/common/oauth2/v2.0/autho
rize?client_id=79297b4d-a602-4206-9b53-9ceb9633e6e0&response_type=token+id_tok
en&redirect_uri=https://subdomain.localhost:9080/auth/callback&scope=user.read
+openid+profile+email&response_mode=fragment&state=12345&nonce=678910


REACT_APP_REDIRECT_URI=http://subdomain.localhost:9080
```

Answer: The Microsoft redirect url is the IP of the server where the sdcoe-transcript project is being deployed. Just changing this will not work as expected, as this should be configured in the Microsoft console. The redirect url should be configured in the Microsoft console and updated in the .env file of the web-client else it will popup with the error message.(Redirect URL Mismatch)

Yes, we need to add the redirect url in the Microsoft console. Yes, you can add the redirect url if you have access to the MS console. The console is already configured with a couple of redirect urls configured. Some of them were used for development purposes only like localhost:9080 and localhost …

QUESTION: Where in the code does authentication occur and which database/tables does authentication information get stored?

Answer: In transcript-api module. This module is concerned with handling login and authentication. The authentication functionality in this module validates whether the user email exists in the users table in the sdcoe database after validation of the token received from Microsoft.

QUESTION: There was concern about PDF rendering differences between Linux and Windows. The technology for the Linux version was an application or library that may no have worked for Windows. Is this still an issue? How was this resolved?

Answer: The puppeteer library is used for generating pdf and the handlebars is used for templating. Both the libraries are compatible with both the operating systems. During my setup process, there was no issue while generating the pdf using the web application and the cli command. I don't think this was the issue, but still if you find issues in this step of the pdf generation add the specific issue and let us know. We will work to fix it.

QUESTION: There is a multiple Composite ODS feature to read Composites from multiple ODSs. How is this feature enabled and where are the details for multiple endpoints stored?

Answer: It is stored in the edfi_ods table. We have added new columns to the table to store clientId, secret and the base url. Simply running a query to insert the new row in the edfi_ods table, you can add the ODS config to the database. Refer to section 5 for adding multiple ODSs.

We can simply run the INSERT query in the database console or any database management tool.

Insert Query

```
INSERT INTO edfi_ods(client_id, client_secret, base_url) VALUES('clientId', 'clientSecret',
'baseURL');
```

Example Query

```
INSERT INTO edfi_ods(client_id, client_secret, base_url) VALUES('KEY/ID', 'SECRET',
'https://youredifiapi.com');
```

QUESTION: There have been demos where the actual Composite endpoints have not been used. Instead, there are local JSON files on the server (we presume) used in processing and testing. How is this configuration changed from ODS to static source data files?

Answer: Simply setting the NODE_ENV to production which is already configured with cron scripts.

```
"setup-cron": "set NODE_ENV=production && babel-node ./src/scripts/cron.js",
 "setup-cron-linux": "NODE_ENV=production babel-node ./src/scripts/cron.js",
```

QUESTION: Where are emails addresses hardwired for testing and set for real usage?

Answer: The transcript-api requires you to send an email in various cases. So, those SMTP configurations should be included in the .env file of the transcript-api. If you want to test the email utils independently you can also configure the .env file of the email-utils.
Refer the following .env section of the transcript-api.

```
SMTP_HOST=...
SMTP_PORT=...
SMTP_AUTH_USER=...
SMTP_AUTH_PASSWORD=...
```

QUESTION: What are the recommended steps to stop the various services if you needed to restart a server running the transcript application? Is this what you would shut down? Are there any others?

```
$ pm2 delete ganache-server
$ pm2 delete transcript-api
$ pm2 delete verification-api
```

Answer: Stopping the Server running in PM2

```
# List the services running
$ pm2 ls

# Stop all the server
$ pm2 delete all

# Stop server by name
$ pm2 delete ganache-server
$ pm2 delete transcript-api
$ pm2 delete verification-api
```

QUESTION: After all of the steps are complete to get the application running (above in this deployment guide), and all is working well, if a Windows Server or Windows 10 Home machine were to be restarted - would these be the steps to restart the application?

```
Run the ganache-cli
        $ cd [path]/sdcoe-transcript/security
        $ pm2 start cli/startGanache.js --name ganache-server

Deploy smart contract
        $ yarn deploy-contract --port 8545

Start the transcript-api server using pm2
        $ cd [path]/sdcoe-transcript/transcript-api
        $ pm2 start dist/index.js --name transcript-api

Start the verification server using pm2
       $ cd [path]/sdcoe-transcript/verification-api
       $ pm2 start dist/index.js --name verification-api
```

Answer: Yes, you need to start the servers running via pm2 again. As all the API and frontend is built already, we can only start the api servers.

Security Utils

```
## start PM2 Server
$ cd security
$  pm2 start cli/startGanache.js --name ganache-server

## Deploy the smart contract
$ yarn deploy-contract --port 8545

## As we are using the same mnemonic, the values remain consistent. If you changed the
mnemonics in the .env file. You need to copy the DID registry address and update the .env
file.

## Set Attribute
$ node ./cli/didFactory.js set-attribute did/pub/RSA/veriKey/base64
"$SDCOE_VERIFIER_PUBLIC_KEY_BASE_64"

## Resolve the DID
$ node ./cli/didFactory.js resolve-did "did:ethr:$BLOCKCHAIN_ACCOUNT_ADDRESS"
```

Note: You need to update the .env file of the transcript-api, verification-api and security module if the mnemonics, and port are changed. Using the same configuration persists the same account info.

Transcript-API

```
# After the build, dist folder contains all modules and dependencies. Now run the index.js file in the dist folder to start the server.
$ cd transcript-api

$ pm2 start dist/index.js --name transcript-api
```

Verification_API

```
# After the build, dist folder contains all modules and dependencies. Now run the index.js file in the dist folder to start the server.
$ cd verification-api

$ pm2 start dist/index.js --name verification-api
```

As, the frontend is already built and mapped with the IIS server. The frontend should work as expected if the IIS server is working.
You need to configure the IIS server to the previously built folder i.e. build in web-client directory and set up the reverse proxy again if the IIS server configuration is removed/deleted.

List the running services

```
$ pm2 ls
```

Delete the services

```
# Stop all the server
$ pm2 delete all

# Stop server by name
$ pm2 delete ganache-server
$ pm2 delete transcript-api
$ pm2 delete verification-api
```

QUESTION: How can one create a backup of the SQL data?

1. Open SQL Server Management Studio Express and connect to the SQL server.
2. Expand **Databases**.
3. Right-click on the database you want to back up, then select **Tasks > Back up**.

This will create the backup file for the database.