Faculty of Arts and Sciences

Department of Computer Science

CMPS 283 – Computer and Information Security

Spring 2021, Dr. W. Hajj

Project Report

For the Group Term Project:

Password Manager

Team Members:

Adam Helal

Aya Nashawati

Youssef Itani

Demo Link:

You can find the demo for the application here: Application Demo

---

Project Idea:

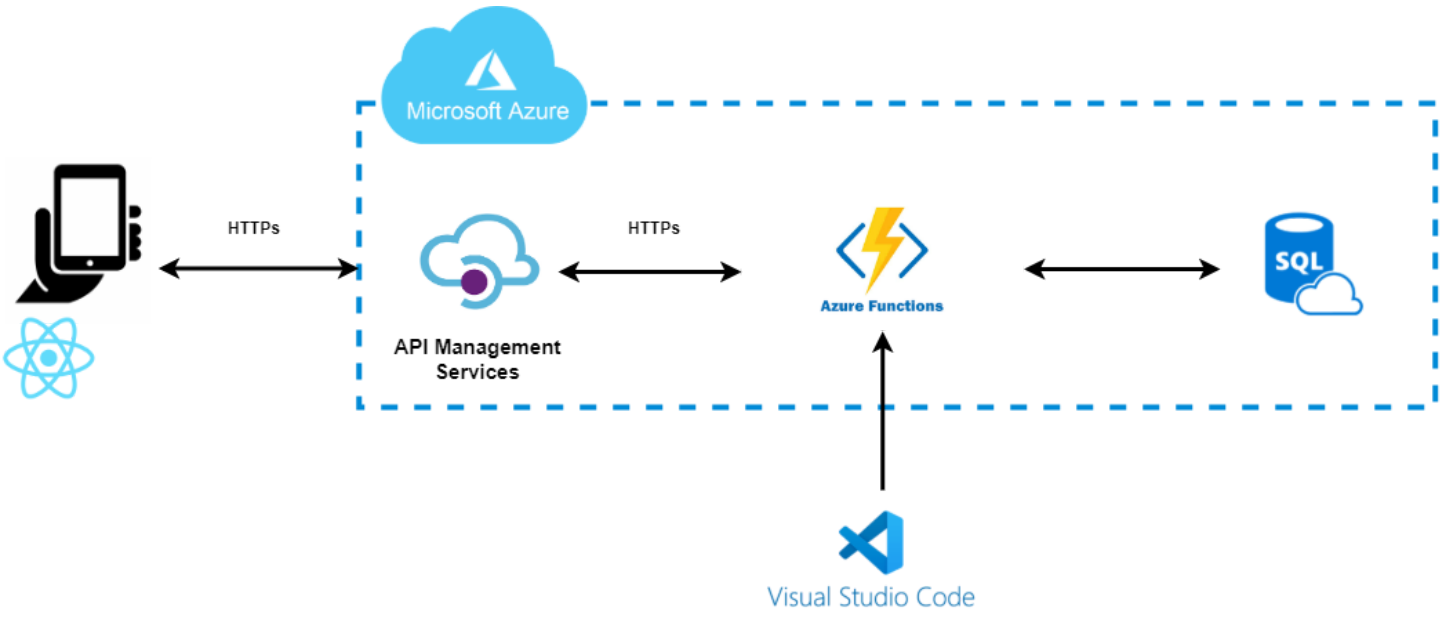
A cloud-based, cross-platform password management mobile application that provides the user with a seamless and user friendly experience whilst maintaining the highest levels of security by enforcing industry standards such as encryption, two factor authentication, password feedback/recommendations and more.

How is it different?

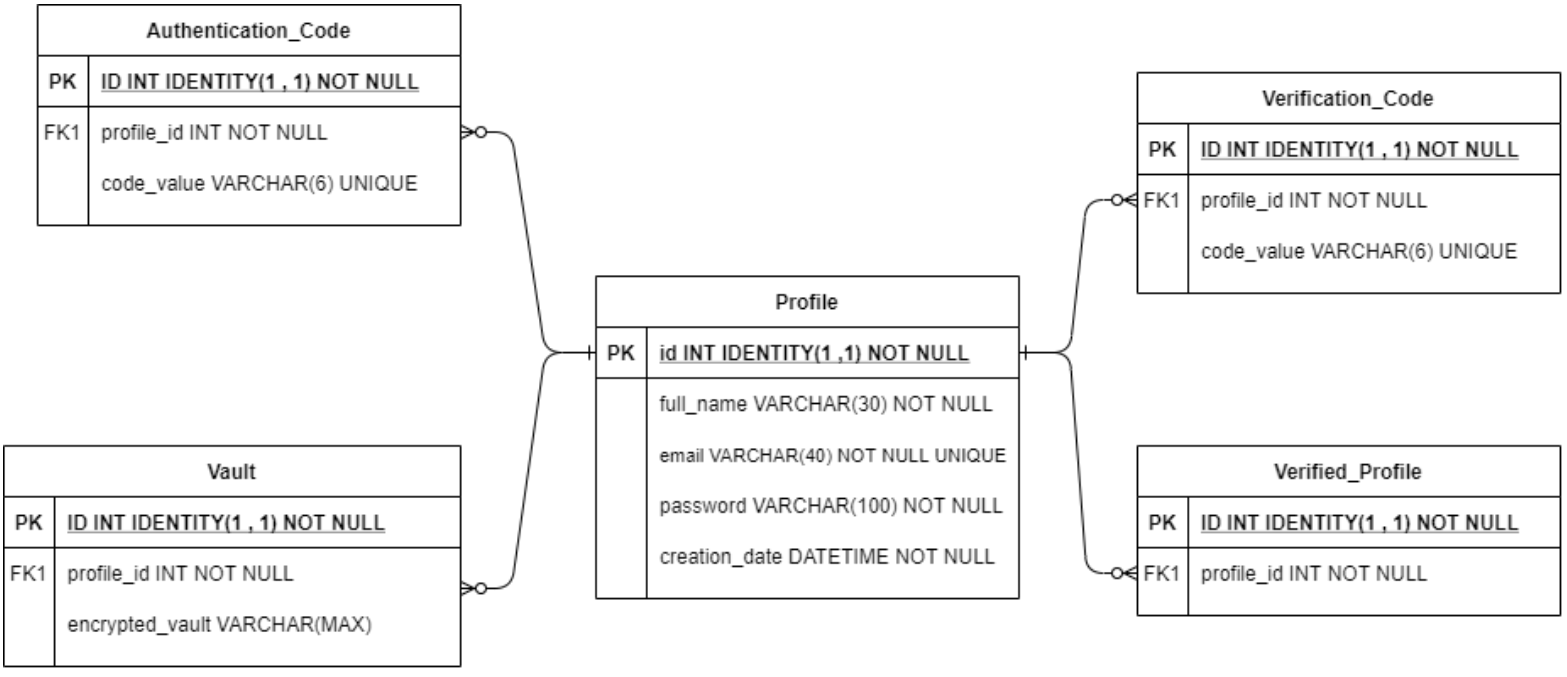
Similar to modern password management tools, our application follows well known industry standards. A common feature that we included was the ability to add, edit and delete passwords. However, we wanted to make our application more secure and informative (helping the user follow best security practices), which is why we implemented the following features:

- We use Two Factor Authentication.
    - Since the user is trusting our application to store their passwords of various websites and applications, we as a result must decrease the chance of a hacker gaining access to such sensitive information.
    - An account for a user will not be created until they input the verification code provided to them.
    - Each time a user tries to login to their account they must provide the new authentication code sent to the provided email.
- Checks password strength.
    - We want to encourage our users to choose strong and secure passwords, which is why we included a strength bar that shows the user the strength of their chosen password.
- Auto generates extremely strong passwords.
    - We included the option for the user to auto-generate a very strong random password. This makes it nearly impossible for the user to reuse the same password for various websites and applications, making it more difficult for hackers to guess the passwords.
- Synchronisation between offline and online modes.
    - Aiming for convenience and a wide user base with varying network connections, we decided to make the app accessible without internet access (providing that the user logs in beforehand). Constant synchronisation is in place to make sure that the user's actions are saved both on the database (when the user is connected to the network) and on the local device itself.

Architecture & Technology Stack:



Entity Relationship Diagram:



Implementation:

Server-Side:

For the server-side, we chose to deploy REST APIs using **Azure functions**, and the **Azure API Gateway**. For this, we have created an API Gateway and imported all the functions created using python, in order to register them to it. Deploying the API Gateway side by side with the API hosted on the cloud, will allow us to optimize the API traffic flow. Moreover, this allows us to add more security by setting policies and restrictions on each function. One of the policies we have added is **limiting the call rate per IP address for a certain time period**. In addition, each function will require the API key in order to access the Azure Resources. By using an API Gateway, we are providing **Availability** to our users, by preventing potential DoS attacks, and lowering the traffic, thus allowing more users to use our application. In addition, we are able to prevent any online password cracking attacks, which include brute force attacks.

By using Azure functions, we are providing **Data Integrity** and **Confidentiality** to our users. By default, clients can connect to function endpoints by using both HTTPS, which uses the SSL/TLS protocol to provide a secure connection that is both encrypted and authenticated.

When it comes to storing the passwords and accounts, all of the account details are present in a list stored in an array of account objects (JSON). This array is encrypted at the client-side using **AES** and the master password (**which only the user is able to generate**) and stored on the Azure SQL database**.** We chose the Azure SQL database, because its newly created databases are **encrypted by default** and the database encryption key is **protected by a built-in server certificate**. Certificate maintenance and rotation are managed by the service and require no input from the user. Moreover, SQL Databases are protected by **firewalls** in Azure. By default, all connections to the server and database are rejected if not authorized.

Client-Side:

For this project, we used the React Native framework managed by Expo, which allows us to create cross-platform applications, benefit from its various libraries, and easily deploy our application.

We have provided several features to assist and ensure the user follows **best security practices**. These features include a graphical password strength checker and an auto-generate password option that generates very strong passwords.

On ***sign up***, the user will register by providing their email and password (which must be a minimum of 8 characters as recommended by *NIST requirements*, and follows the **comprehensive** password policy with the exception of dictionary words). The passwords in both ***sign up*** and ***login*** uses **multiple encryption** where the password is hashed twice using different hashing algorithms. After signing up, they receive an email with a **verification code**; this allows us to verify that they are who they claim to be. Note that the user will not be able to use the application unless they are verified, since the functions only process requests *if and only if* the user is verified.

While ***logging in***, a vault key will be generated using the combination of the password and email, and hashed using **PBKDF2 (using 1000 iterations).** In order to add more security to one's account, Two-Factor Authentication is enforced. Each time the user logins in, they must input the new authentication code sent to the email they provided. If
After logging in, the user will retrieve their encrypted vault from the SQL database and decrypt it on their end using the master password (**cryptographic function used is AES**). Keep in mind that neither the server nor the phone will know what the master password is, since we will ask the user to enter their password every time they open the application (it is not stored anywhere).

However, if the user didn't log out but simply closed the app, then once they open the app again they will be greeted with a page that asks for their master password (instead of logging in again). This is also the case when the user is using the app and the internet connection cuts off.

Moreover, full synchronisation is in effect to ensure that if there is no internet connection, the user can use the application(with the latest passwords) normally, **only if they did not log out**. When they regain a network connection , the vault stored on the database will be updated with the user's newly added, edited and/or deleted passwords.

Moreover, the user also has the option to permanently delete their account. However, for verification purposes the user can only do so if they retype the email that they created their account with. This feature, along with SignUp and Login, are only available in the presence of internet connection.

Guide on how to install and run the Project:

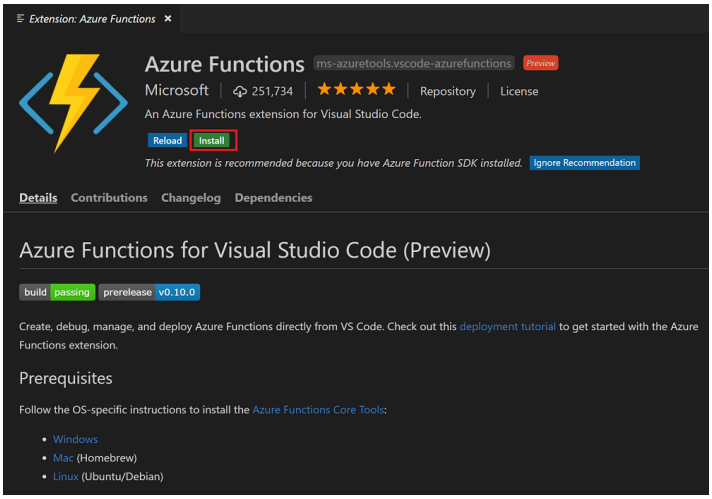
Backend Implementation

For this project, we used **Microsoft Azure**. We were able to access its service thanks to the **student benefits** that Github provides for university students. We have created an Azure SQL database , and we will be using it for this project

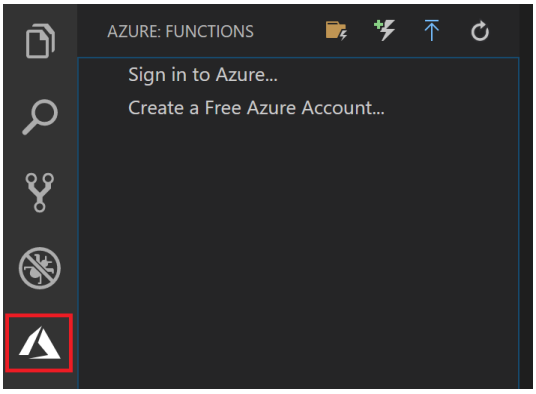
Setting up the Serverless Functions

Install the Azure Functions extension

1. You can use the Azure Functions extension to create and test functions and deploy them to Azure.



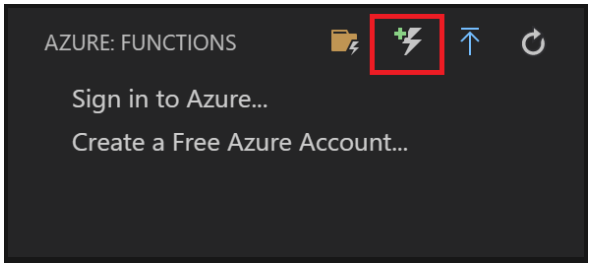
2. After installation, select the Azure icon on the Activity bar. You should see an Azure Functions area in the SideBar.



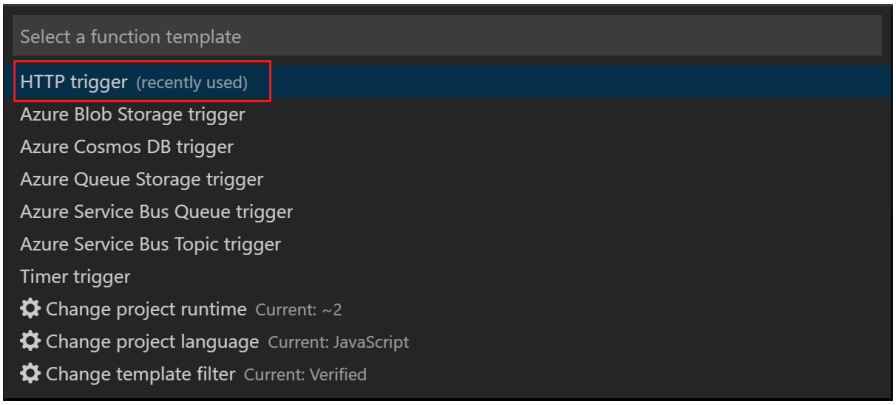
Create An Azure Function Project

The Functions extension lets you create a function app project, along with your first function. The following steps show how to create an HTTP-triggered function in a new Functions project. **HTTP trigger** is the simplest function trigger template to demonstrate.

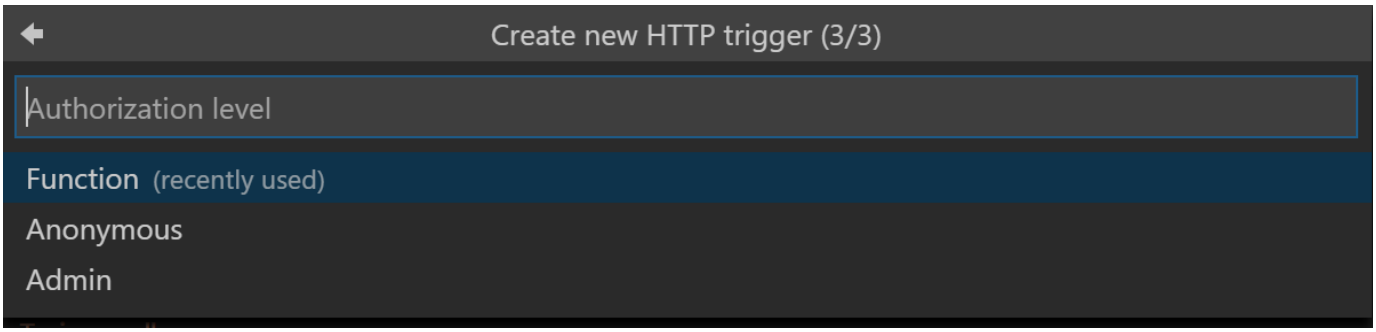
1. From Azure: Functions, select the Create Function icon:



2. Select the folder for your function app project, and then Select a language for your function project.
3. Select the HTTP trigger function template, or you can select Skip for now to create a project without a function. You can always **add a function to your project** later.



4. Type HttpExample for the function name and select Enter, and then select Function authorization. This authorization level requires you to provide a **function key** when you call the function endpoint.



Running Functions Locally (Using our functions)

1. Open **Windows PowerShell** and **run as administrator**
2. Enter the following command **`Set-ExecutionPolicy RemoteSigned`** and press **A**

```
PS C:\WINDOWS\system32> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"):
```

3. After doing this, go to VS Code, and run the **debugger**.
4. When done, go back to the **Windows PowerShell** and enther **`Set-ExecutionPolicy Restricted`** and press **A**

```
S C:\WINDOWS\system32> Set-ExecutionPolicy Restricted

xecution Policy Change
he execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
ou to the security risks described in the about_Execution_Policies help topic at
ttps:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): A
```

Front-End Implementation

For the frontend implementation, we used **React Native Framework** managed by the **Expo Platform**, which allows us to create cross-platform applications, benefit from its various libraries, and easily deploy our application.

Prefered Method to run: using the provided APK

For simplicity we already created the application, which allows you to run it on any Android phone at any time.

An APK stands for Android Package Kit which is the package file format used by the Android operating system, and a number of other Android-based operating systems for distribution and installation of mobile apps.

To use the app, simply:

- Find the APK in the provided zip folder and download it.
- Install the APK.
- Run the App and enjoy!

Other Method: using Expo

Expo is a set of tools and a framework that sits on top of react native, and helps in testing the app on mobiles and web browsers over-the-air (OTA).

To install and setup the development environment for Expo and react native, kindly watch the video below:
Installing Expo

How to run and install the project on your computer:

```
cd {project name}
npm install
expo start
```

Example (please make sure to configure your execution policy to remotesigned when using the command expo as indicated in the end of the backend guide above):

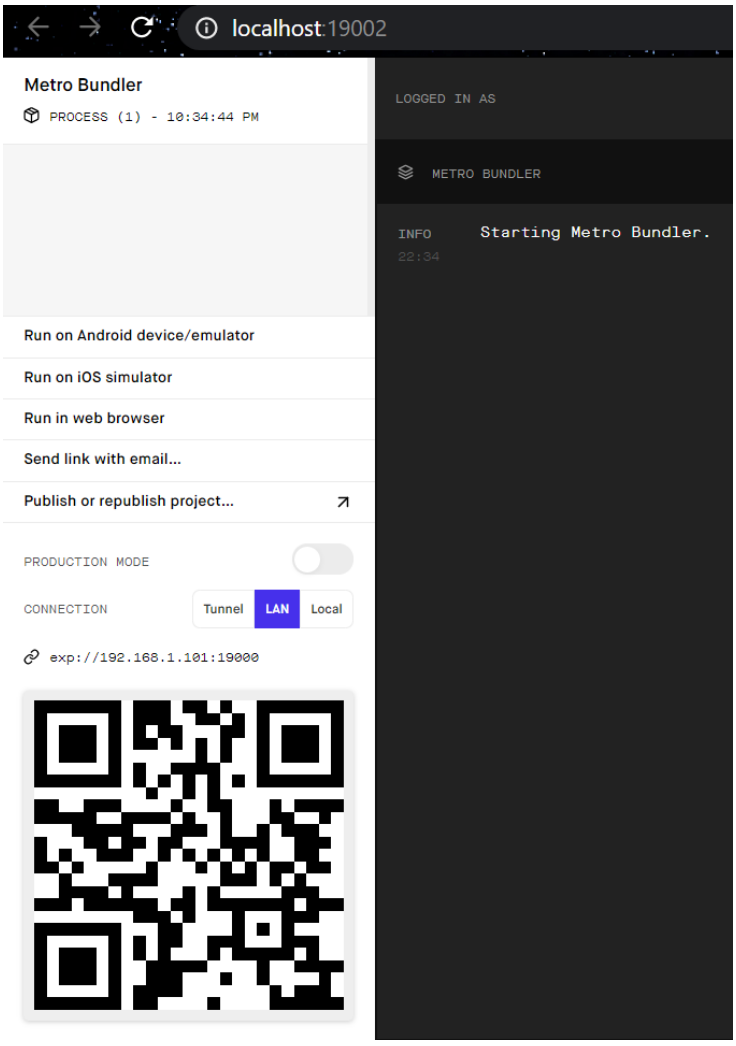
```
cd PasswordManager
C:\PasswordManager>npm install
C:\PasswordManager>expo start
```

Notes on the React Native folders in the project:

- The assets folder contains all the images we used.
- Node modules folder contains all the different packages and dependencies that we need in the project.

- App.js is the file that kickstarts the app.
    - App is the Root component of the Application.
- App.json contains information about the project.
    - Name.
    - Platforms.
    - Icons.
- Babel.config.js
    - Configuring how babel works with the project.
    - Bable is the compiler that allows us to use modern js features.
- Package file allows us to track different dependencies in the project and scripts.

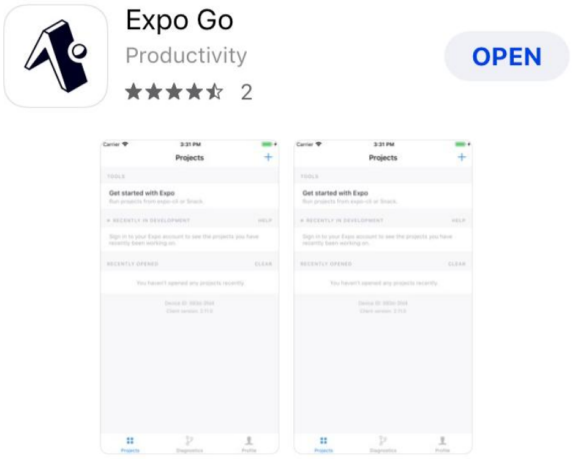
After running “expo start” the following will appear:



Note: If you want to test the app offline via expo, then you'll have to press the LAN button in the image above then connect your phone via usb and press "Run on Android device/emulator" (as this option only works for android phones).

How to use Expo on your mobile phone:

You must first download the Expo application.



After doing so, you must then scan the QR code provided after running "expo start".
Upon pressing the resulting notification, you will be taken to the Expo app where our Password Manager application will be loaded.