Password manager

* Ensure only application can communicate with the server.
* Ensure that the connection between application, server and database is secure (https, API keys, client certificates, authorize access)
* use these for server (might not be needed)
  + Creating the salt: <https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.randomnumbergenerator?view=net-8.0>
  + Hashing: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.passwordhasher-1?view=aspnetcore-8.0>
* use these for application:
  + Creating salt: <https://doc.qt.io/qt-6/qrandomgenerator.html#securelySeeded>
  + Hashing: <https://doc.qt.io/qt-5/qcryptographichash.html>
* ElephantSQL if wanting to host these on outside service. Secure?
* dockerize server and database, docker network
* validate and sanitize the input coming from the application
* rate limiting?
* Https to transfer data between server and application

steps of storing password

1. the derived key is generated on the client side using the master password and possibly another salt (different from the one used for hashing the master password)
2. normal password is encrypted with derived key as encryption key
3. encrypted password is sent to server via https
4. password is then stored in the database

steps of verifying user

1. users master password is salted and hashed and is stored in the database
2. user logs in with the master password
3. application sents the username to the server to retrieve the salt.
4. application then salts and hashes the master password
5. that is then sent to server to compare
6. if it’s correct then everything is ok

In the end

1. users master password is salted and hashed and is stored in the database
2. user logs in with the master password
3. application sents the username to the server to retrieve the salt.
4. application then salts and hashes the master password
5. that is then sent to server to compare
6. if it’s correct then everything is ok
7. the derived key is generated on the client side using the master password and possibly another salt (different from the one used for hashing the master password)
8. salt for derived key stored in the database
9. normal password is encrypted with derived key as encryption key
10. encrypted password is sent to server via https
11. password is then stored in the database

1. Client to Server: Clients must be authenticated and authorized by IdentityServer before they can access your server.

2. Server to Database: Only your server can establish a connection with your database, as controlled by the pg\_hba.conf configuration.

Create certificates

1. dotnet dev-certs https -ep $env:USERPROFILE\.aspnet\https\server.pfx -p 1234
2. dotnet dev-certs https –trust