SECUR-O-TECK API

1. API

   An API (application programming interface) is a set of communication protocols that enable different software components to communicate with each other. In the context of distributed systems/applications, Web APIs facilitate the transmission of requests and responses. This server is stateless because it does not store any previous requests. Each request contains the all the information it needs. This is good because the server is expected to work with an arbitrary number of unknown clients like the majority of the web. It is also good for managing thousands of simultaneous requests/sessions that would otherwise be too intensive if stateful (lots of state to store and manage, high RAM usage, hard to know when to end a sessions, etc.). On the other hand, stateful servers are useful when working with a tiny number of known clients as each session is customized to each client. It is possible to create a stateful server for this API, however ROA would not result in a great amount of efficiency. The API manages incoming requests via route mapping.

2. Route Mapping

   Upon receiving an HTTP request, the server attempts to match the controller with the one provided in the URI. If no match is made, a 404 Error Not Found error is sent to the client. The route mapping has been changed from the conventional api/{controller}/{id} to api/{controller}/{action}/{id}. This allows the user to call actions from inside the controller and have the id passed through the headers, thus eliminating the need to pass it in the URI. The action identifier gets matched with the correct method action name and the id (if it exists) gets passed through the headers for the server to process it.

3. HTTP Requests

   1) GET

      Retrieves information from the database. Must be idempotent, as in for the same parameters, the results are the same irrespective of the number of requests. Side effects are possible but should not be operation critical as they are not expected by the user.

   2) POST

      Requests that the asset/resource/object in the URI execute an operation with the entity provided. Used to create an entity, in this case a new user, or update/modify an entity, for example promoting a user from user to admin.

   3) DELETE

      Request the deletion of an entity, either taking effect immediately or asynchronously.

4. API Key

For a stateless server like this, passing the API key is a good option for identifying users because it’s fast however not secure, because the id is being transferred in plain text and could be compromised via Man in the Middle Attack. In the real world the id and API key would be encrypted, albeit sacrificing some efficiency. The id would be encrypted with the server’s public key. The asymmetric key is the first thing we need to use because the server can send the public key and using that we can encrypt the API key and the symmetric key and send it to the server. Server decrypts the id with the private key and checks the API key to see if we exist on the database. It then encrypts with the symmetric key and sends back an encrypted response. The client can send the messages with symmetric encryption which is much faster than asymmetric encryption. Asymmetric encryption is good for exchanging data at beginning and afterwards we can use symmetric encryption for further communication.

5. Encryption

1) RSA

Key Generation:

1. Two distinct primes p and q
2. $n = p \times q$
3. $\lambda(n) = \mathrm{lcm}(\varphi(p), \varphi(q)) = \mathrm{lcm}(p - 1, q - 1)$
4. Integer e where $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; i.e., e and $\lambda(n)$
5. d as $d \equiv e-1 \pmod{\lambda(n)}$

Encryption:

$$c \equiv m^e (mod\ n)$$

Decryption:

$$c^d \equiv (m^e)^d \equiv m(mod\ n)$$

2) AES

1. Key expansion
2. AddRoundKey
3. SubBytes
4. ShiftRows
5. MixColumns
6. AddRoundKey
7. SubBytes

8. ShiftRows
9. AddRoundKey

6. Entity Framework

An open source object relational mapping framework that allows developers to work with data without worrying about manually setting up the database tables. This enables a high level of abstraction and reduces the code required to build data oriented, distributed applications.
The Entity Framework provides three workflows for creating an entity model:

1) Database First

   The developer designs the tables first and then Entity Framework generates the domain classes. This is the traditional approach that has been used for many years. It has the advantage of being simple to create through a GUI and mapping of key-foreign key relationships can be done without code. Also good for large, data heavy applications.

2) Code First

   The domain classes are created in C# first and then EF generates or updates existing database tables via an EF feature called context that manages the interaction between classes and database tables. This happens at runtime and exists in the memory so the developer will never see this model. If the model changes and becomes misaligned with the database, a migration is performed to update the database tables. Automatic migrations can do that for us, however at the risk of data loss.

3) Model First

   UML diagrams are created in a visual designer to model the classes and their associations.

Code First is chosen for this database because it is suitable for small applications, database version control, and because database tables can be defined from the business objects.

8. Tasks

All tasks were completed to their full extent as per the ACW specification.

Task	Problems	Solutions
6	Logs were not being transported to archive logs properly. User_ApiKey was null in logs.	Moved logging method call to the top of APIAuthorisationHandler, set the for loop iterator to -1 at the end of the user.Logs iteration, and decremented count by 1.

13	Passing User through method parameters threw null reference error.	Passed ApiKey string instead and retrieved the user via query where ApiKey matched.
14	AddFifty crashed on the server.	Referenced MSDN AESCryptoServiceProvider. Fixed name in the parameters so its matches the one in json.