Firstly let me tell you about what I am doing at my current company. My company is printing company. Regular customers of my company are coming from banking industry and insurance industry. So they buy poster, brochure and business card from my company. To create those product, they use our web application systems to upload required data so that we can generate poster or brochure using their data. As they are from financial industry, they need very strong security for those data they gave to us. That is the reason why I involved in the security area.

So, firstly, let me tell you about how I manage security for customer data file. Some customer upload excel data file and some customer upload zip file, but whatever it is, those files are password protected. So our system have to protect user’s password regarding to their data file. If our system keep password as plain text, then any hacker can get it easily. So I have to encrypt to convert to ciphertext.

**Encryption on user’s credential**

So I want to tell a little about how encryption work based on my experience. To make it encrypt my system need to generate secret key so that it can be used to decrypt or to read the ciphertext. It is very simple way for encryption. But It has the problem. Let’s say that user key in the same password for several times, then cipher text will always be the same which is not good. It can make it easy to guess what will be the password. Best practices is to make it harder to guess. So I used IV – Initialization vector which is nothing but a group of random numbers. So using IV can generate different ciphertext even if user key in the same password for multiple time which make it difficult to guess. It is good practices.

**Clean up data regularly**

But I still have another problem, my customers don’t want to keep their data file in our database forever because it is confidential. So they gave to my system certain period such as 14 days or 30 days. Based on that certain period I implemented scheduler service which delete data files regularly. It is another security best practice.

**IPs based authentication**

But I still have another problem, my customers don’t want everyone can access to the system except the request(s) which come from specific IPs. Because they are on the financial industry. So I have to add additional function which will be checked every request and their IPs. If the requests are come from the whitelisted IPs, my system allowed them to go on. Otherwise, they never can login to my system. It is also another security best practice.

**Protect XSS and SQL Injection attacks**

But I still have another problem, my customers want to make sure that our system cannot be attacked by XSS and sql injection attacks. So I implemented three things to protect those attacks. The first thing is I create a function which encode whatever user key in to my system and also encode whatever data want to display to user. The second is checking every user input using regular expression patterns. For example, user key in for postal code, then it should not be included any special characters. Using regular expression, I can check whatever user key in to my system. The last thing I used is linq to entity framework to protect sql injection.

Those are the overall experiences from my current company. If you don’t mind let me explain about how my web api framework working in overview. Because I afraid that you will get boring.

The first thing I would like to explain is user management in my web framework. There are only two things in user management for every application. They are authentication and authorization.

Authentication is nothing but just checking the users whether they are allowed to use the system or not. To implement the authentication in my framework, I had two ways to choose. The first way is cookie based authentication and second way is token based authentication. Cookie based is traditional way and token based is modern way. Let me explain a little bit about what are the differences.

**Cookie Based Authentication or Statefull mechanism**

The way cookie based auth working is that user give user name and password to my asp.net application server. Application server verify credential whether it is legitimate one. If it is legitimate one, application server generate session object which included session key and send these session key to the user who is the owner of these user name and password. Then user browser keep these session key in cookie. Then whenever that user make the request to this server, browser send his session key to application server. Then application server verify his session key whether it is legitimate one comparing with its associated session object which already keep in application server memory. That way is also called statefull mechanism. According to my experience, I don’t like statefull mechanism because it depand on specific application server memory. If these application server down, your session will be lost. It mean that you cannot use the application any more.

**Token based Authentication or Stateless mechanism**

Another way is token based authentication. It has no session object on web application server. It means that it can change server to server physically without having any errors. It means that it is more scalable. User send user and password to server. Server generate token key and signed digitally for that user. Then user receive these token and send it back to server whenever he made the requests. Server verify the token and process the requests. It means that you can change server to server without worrying about losing session object because it is stateless. In my web framework, I use that practices.

**Handling application time out**

I use the stateless mechanism. It means that I have no state. No state means no more session. So how I control time out for my application is quite challenging. Authentication token that I used has expiry logic in its own way. But it is not work for some kind of situations. For example, one of the users login to my application and received authentication token which will expire in 15 minutes. He is doing his job using my application without going to anywhere. It means that he is active for 15 minutes. But according to expiry time he will be automatically logged out after 15 minutes no matter his current status is active or not. It is very troublesome. So I implement application time out in my own way. Using javascript, I can know if current user are active or not for a specific time. If the user are inactive for specific time, then javascript send message to server to delete token. In this way I managed the application time out.

**SSO feature in E-commerce**

Another thing which I want to explain is about my E-commerce application experience. I created e-commerce system for one of the insurance company in Singapore. Insurance agents login to my system and they buy insurance brochure from my system using Paypal. After creating this system, I found the problem. My customer already has java backend system which mean they already have login page in their backend system. That is why they don’t want to login twice since they already login to their backend system. So I needed to implement SSO single sign in feature. So their system give encrypted xml stream to my application via https. Then I needed to decrypt and get the user’s credential from this. In this way our customer no need to login multiple times.

**When web application should not be used**

The final thing based on my own experiences, I want to explain when we should not use the web application. There is some situation we should not use the web application. Whenever we need to execute a program which will take time more than we can think of, then we should not use web application. Because every web application has time out. That application time out can happen before you finish execution the program. It is very annoying. In such kind of situation we should use Messaging Queue service and Window Service. By using Messaging Queue and Window Service we can handle such kind of situation.