**ThreeAmigos Corp**

The project includes:

* A Web application [Managing Events] -- **Web App**
* A **Web Service** – Manages Venues, Availability and Reservations
* A **Web API & Web Application** – Manages Catering as both an API and a Web Server

The **WebAPI template** starts with a Controller class that will allow you to respond to RESTful requests at the /api/Values endpoint.

The **Web Application template** provides an MVC framework enabled project with some Razor views, the bootstrap CSS framework and jQuery library installed.

**Security Features**

**1. Cross-Site Request Forgery (XSRF/CSRF) attacks**

Cross-site request forgery is an attack against web apps whereby a malicious web app influences th e interaction between a client browser and a web app that trussts that browser.

This attack is possible since web browsers send some types of authentication tokens automatically with every request to a website.

The attack is simply taking advantage of the user's previously authenticated session.

CSRF attacks are possible against web apps that use cookies for authentication because:

* Browsers store cookies issued by a web app.
* Stored cookies include session cookies for authenticated users.
* Browsers send all of the cookies associated with a domain to the web app every request regardless of how the request to app was generated within the browser.

To guard against CSRF vulnerabilities:

* Sign out of web apps when finished using them.
* Clear browser cookies periodically

ASP.NET Core implements antiforgeyr using ASP.NET Core Data Protection and the data protection stack has to be configured to work in a server farm.

In this project this attack is taken care of by:

* Explicitly disabling antiforgery tokens with the **asp-antiforgery** attribute
* The form elements are opted-out of Tag Helpers by using the Tag Helper **! opt-out symbol**

**2. Cross-Site Scripting (XSS) in ASP.NET Core**

Cross-Site Scripting (XSS) is a security vulnerability which enables an attacker to place client side scripts (usually JavaScript) into web pages.

XSS vulnerabilities generally occur when an application takes user input and outputs it to a page without validating, encoding or escaping it.

The Razor engine used in MVC automatically encodes all output sourced from variables.

The engine uses HTML attribute encoding rules whenever the **@** directive is used.

Validation is also a useful tool in limiting XSS attacks. Untrusted input should be encoded before output, so matter what validation or sanitization has been performed.

**Endpoints**

The following are the Catering Web Service endpoints\

* Food Bookings API end points
* /api/foodbookings
* /api/menus
* /api/fooditems
* Web Server endpoints [Food Items]
* /FoodItems/Details/{id}
* /FoodItems/Create
* /FoodItems/Create
* /FoodItems/Edit/{id}
* /FoodItems/Edit/{id}
* /FoodItems/Delete/{id}
* /FoodItems/Delete/{id}
* Web Server endpoints [Menu]
* /menus
* /Menus/ViewFoodItems/{id}
* /Menus/Details/{id}
* /Menus/Create
* /Menus/Create
* /Menus/Edit/{id}
* /Menus/Edit/{id}
* /Menus/Delete/{id}
* /Menus/Delete