WEB APPS ICA Report:

# Completeness:

The integrity and thoroughness of the ICA (Integrated Catering Application) can be evaluated as ranging from Minimal-Viable to moderate. While a substantial portion of the ICA remains incomplete, particularly in the areas of venue booking and staffing positions, I have decided to showcase my proficiency by addressing the essential requirements through the use of MVC (Model-View-Controller) architecture, HTML, and JavaScript.

One notable example is the Menu editor and view, which exhibits commendable features. I have implemented generic API classes to load information into the MVC controller, facilitating the seamless integration of data. The view is then constructed, incorporating concealed details about the menus. In this process, the browser takes charge, executing ajax calls to complete the remaining aspects of the view. This approach not only demonstrates my expertise in essential technologies but also highlights a strategic and efficient method for handling dynamic content within the application.

# Working Practices:

The main way I write software is with white box testing. I write some code, hit run, test for bugs and repeat.

# Security features:

Given more time, I would enhance the security features by implementing more robust measures:

API Origin Restriction: I would tighten the security of the APIs by restricting their access only to requests originating from the event server. This approach aims to bolster the overall security of the API infrastructure, making it more challenging for potential abuse or susceptibility to denial-of-service attacks. By narrowing down the allowable sources to the event server's origin, we create a more controlled and secure environment for API interactions.

Enhanced Authentication Mechanisms: Implementing advanced authentication methods, such as token-based authentication or OAuth, can significantly fortify the overall security posture. Strengthening the authentication process ensures that only authorized entities can access and interact with the APIs, reducing the risk of unauthorized access or data breaches.

Data Encryption: Employing strong encryption protocols, such as TLS (Transport Layer Security), for data transmission ensures that sensitive information remains confidential during communication between the server and clients. This added layer of encryption safeguards against potential eavesdropping and data interception, contributing to a more secure system.

Input Validation and Sanitization: Implementing thorough input validation and sanitization mechanisms helps prevent common security vulnerabilities, such as SQL injection or cross-site scripting (XSS). By validating and sanitizing user inputs before processing, we can mitigate the risks associated with malicious input manipulation.

Regular Security Audits and Penetration Testing: Conducting periodic security audits and penetration testing on the application can identify potential vulnerabilities and weaknesses in the system. This proactive approach allows for timely mitigation of security risks and ensures the continuous improvement of the application's overall security posture.

Monitoring and Logging: Implementing comprehensive monitoring and logging mechanisms helps detect and respond to suspicious activities in real-time. By closely monitoring system logs, administrators can identify and investigate any anomalous behavior, enabling swift response to potential security incidents.