**To secure our clients' login credentials, we need to consider the possible attacks and act accordingly. We will also need to consider how to secure this credential when it is at rest and in motion.**

TLDR:

The user’s password must be transferred and stored as a one-way salted hash with a slow hashing function. This will take care of securing login credential data in motion and at rest.

We can choose the following hashing functions: bcrypt(15 scalar), scrypt, argon2.

To protect against Injections, our system must detect and limit the use of certain characters that can be used for password injections.

Single quote (')

Double quote (")

Semicolon (;)

Backslash (\)

Comment symbols (--, /\*, \*/)

Equal sign (=)

Parentheses ((, ))

Greater than and less than (>, <)

If needed, these are some additional measures to consider:  
**Transport Layer Security (TLS)**: Encrypt data in motion to protect against eavesdropping (HTTPS).

**Rate Limiting**: Protect against brute-force attacks by limiting login attempts.

**Two-Factor Authentication (2FA)**: Add an extra layer of security beyond just the password.

**Session Management**: Secure sessions using cookies with secure, HTTP-only, and SameSite flags.

**Strong Input Validation**: Validate all user inputs, especially for usernames and email fields.

**Logging and Monitoring**: Detect and respond to potential security breaches.

**Explanation and notes:**

Due to the fact that the login prevents unauthorized access to sensitive financial food truck information, we need to ensure that the passwords are properly secured.

This means that we must secure the authentication process as well as the storage of login credentials (securing data in motion and at rest).

By securing data in motion, we can ensure that attackers who are conducting eavesdropping or man-in-the-middle attacks will not be successful. This can be done by encrypting the data so that it is unintelligible to the attacker.

Securing data at rest follows a similar principle. Compromised databases can lead to password leaks if passwords are stored in plain text. This would be detrimental to our clients. Thus we need a way to store passwords that is resistant to data breaches. Encryption would be one way of solving this problem as well.

Both securing password data in motion and data at rest follow the same principle of making the password unintelligible to the attacker. Encryption is one method of doing this but is insufficient.

The reason behind this is that encryption can be decrypted with a decryption key. If our system is compromised, it does not take much additional effort for an attacker to find a decryption key used within our system.

Thus we should instead strive to not store or transfer passwords at all. If the passwords were hashed using a one-way hashing function, we would have less of a problem with worrying about the attackers decrypting our passwords.

However, attackers can also utilize rainbow tables to reverse engineer the plain text password from the hashed password. The reason this works is because many users like to use predictable passwords. Thus, if we have a large table with popular passwords and their hashed mapping, we can match the hashed values to get the initial passwords.

One way to protect against this is by using salting. If we salt the passwords (add a randomly generated string) to the password before hashing it, it would render a rainbow table useless as the original prehashed passwords are now all different to the list of popular passwords in the rainbow tables. Nonetheless, the attacker can still use a dictionary attack (the principle of a rainbow table attack) by salting their own popular passwords and generating their own dictionary of plain passwords and hashed values.

A way to mitigate this attack is to use a slower hashing function (bcrypt, scrypt,argon2) instead of the faster ones (like SHA-1, MD5, SHA-256). This would make generating the rainbow tables and dictionaries much less efficient and resource-expensive. Thus deterring the attacker from attacking our system.

Short helpful videos:

<https://www.youtube.com/watch?v=qgpsIBLvrGY>

<https://www.youtube.com/watch?v=zt8Cocdy15c>

Furthermore, to prevent unauthorized authentication through injection attacks, we can prevent the use of certain characters that are necessary for injection attacks.

Single quote (')

Double quote (")

Semicolon (;)

Backslash (\)

Comment symbols (--, /\*, \*/)

Equal sign (=)

Parentheses ((, ))

Greater than and less than (>, <)

Here are some additional security measures to consider:

**Transport Layer Security (TLS)**: Encrypt data in motion to protect against eavesdropping (HTTPS).

**Rate Limiting**: Protect against brute-force attacks by limiting login attempts.

**Two-Factor Authentication (2FA)**: Add an extra layer of security beyond just the password.

**Session Management**: Secure sessions using cookies with secure, HTTP-only, and SameSite flags.

**Strong Input Validation**: Validate all user inputs, especially for usernames and email fields.

**Logging and Monitoring**: Detect and respond to potential security breaches.