.NET Suggestions Guide

Exception Handling and Logging

- **Avoid null reference error:**
  - If you are taking any list of instances in your class as a property, then assign an empty array in the class itself to avoid a null reference error.
  - Always check for a null pointer before accessing any property/method of individual object.
- **Avoid array out of bound:** Always check if the array index is not out of bound while accessing any random array element.
- **Audit logs:** Keep Audit log of important entities data changes i.e., Financial account updates.
  For example, when any changes are made in payment or invoice entry, existing values must be saved as audit in the log table.
- **Database logs:** Implement Log4net for database or flat file logging. Database logging is recommended.
- **Try-catch block:** Avoid adding a try-catch block in every method to handle unexpected exceptions this will make your code look messy. Try to identify possible exceptions in an early stage of development. Display a proper message to the end user for all errors and exceptions.

Sensitive Information and Unauthorized access

- **Encryption:** Encrypt sensitive information before keeping them in config or data store.
  For example, The server account password, SMTP details, etc.
- **Avoid cookies for sensitive information:** Don't store any sensitive information in cookies like passwords or usernames directly. Either encrypt values before storing or make use of Session.
- **Avoid unauthorized access:** Block unauthorized access managed via Fiddler / Postman by computing a hash of payload using a solid Hashing scheme (e.g., HMAC-SHA256) on the frontend side and then on the backend side. Before processing, the request computes the hash again on the payload excluding hash and verifies the computed hash and receives hash from the frontend side; If the hash doesn't match block access with 401 unauthorized requests.
- **Use of Anti-forgery token:** Prevents application from XSS attacks in MVC.
  Reference Link

What's better to use with string?

- **StringBuilder over String:** When using concatenation, please use StringBuilder in place of String when you are making heavy string concatenation operations.
  Use $ as per the latest practices string s = "This is result: {test_var} which can be used at {test_var2}";

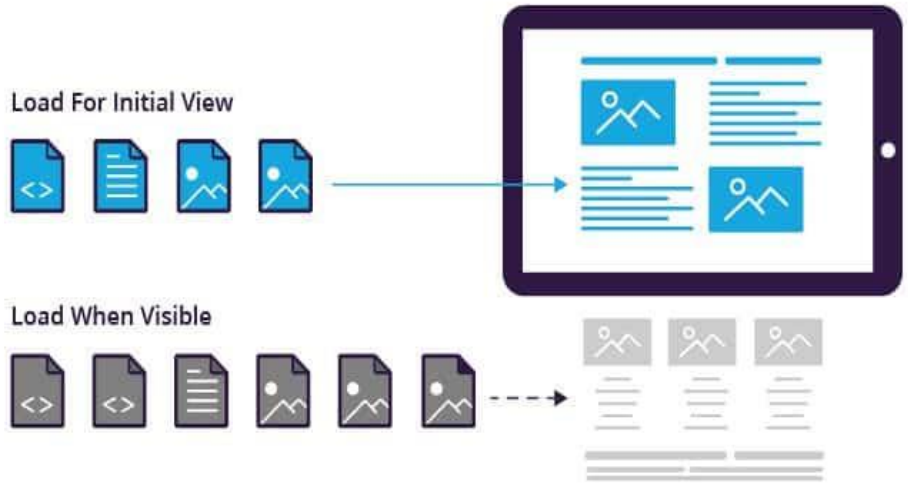
- **IsNullOrWhiteSpace over IsNullOrEmpty:** IsNullOrEmpty checks if there's at least one character, while IsNullOrWhiteSpace checks every character until it finds a non-white-space character.

- **"===" over "==":** Always use "===" instead of "==" when you know the data type of both [LHS and RHS]

Working with Entity Framework

- **Loading:** Keep lazy loading off in Entity framework. Use Explicit loading if it is required.

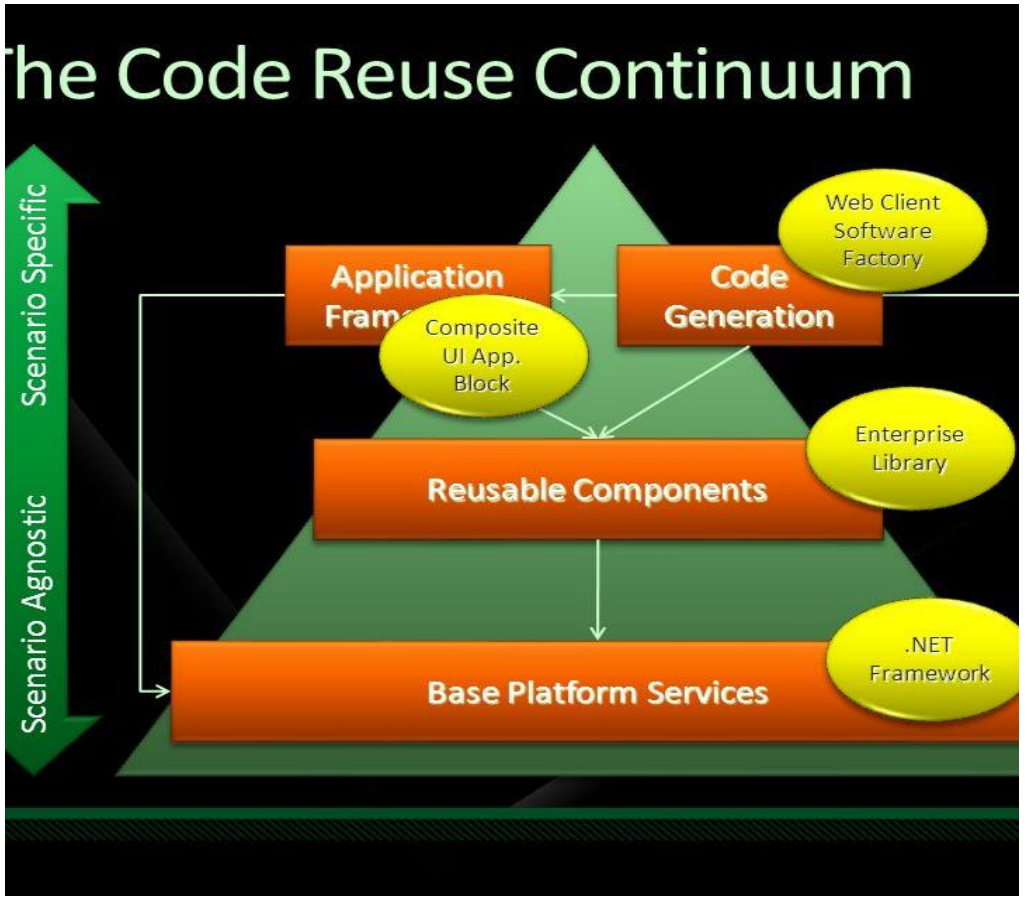
- **ToList():** Do not write the Entity framework ToList() method in a loop. This will fire a query for each iteration and will have an impact on the performance.
- **FirstOrDefault/SingleOrDefault:** Avoid using Where() + FirstOrDefault()/SingleOrDefault(). You can directly use FirstOrDefault()/SingleOrDefault() with predicate.
  - For example: _context.Employee.Where(e => e.id == 1).FirstOrDefault(); is equivalent to _context.Employee.FirstOrDefault(e => e.id == 1);
- **Count():** Do not use Count() > 0 condition in your code. Use Any() instead.



Code Reusability, Cleanup, and Formatting

- **Cleanup and Formatting:** Use lint in the project to make code clean and formatted throughout the project.

- **Reusability:**
  - Have common/shared files for all the static assignments.
  - Keep date and currency calculations in common file and re-use in entire project.



Cross-Origin Resource Sharing (CORS) API:

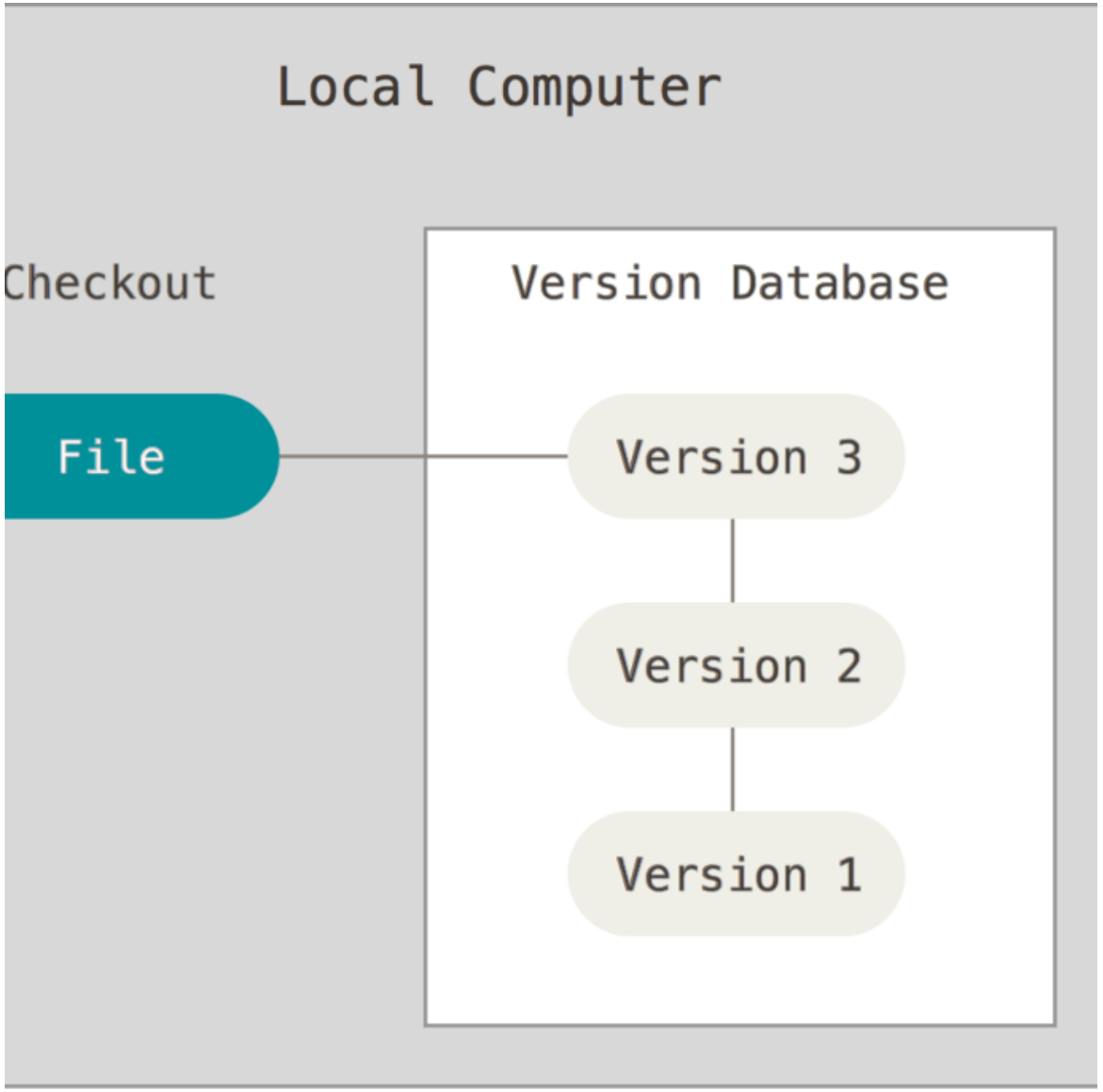
- Using CORS, a server can explicitly allow some cross-origin requests while rejecting others. CORS is safer and more flexible.
  Reference Link
- To initiate a cross-origin request, a browser sends the request with an `Origin: <domain>` HTTP header, where `<domain>` is the domain that served the page. In response, the server sends `Access-Control-Allow-Origin: <domain>`, where `<domain>` is either a list of specific domains or a wildcard to allow all domains.
- Avoid using Access-Control-Allow-Origin: * to allow all domains.
- Instead use 'Access-Control-Allow-Origin: https://www.example.com http://www.example.com https://s0.2mdn.net http://s0.2mdn.net https://static.doubleclick.net' to allow list of specific domains

Few Important Add-ons

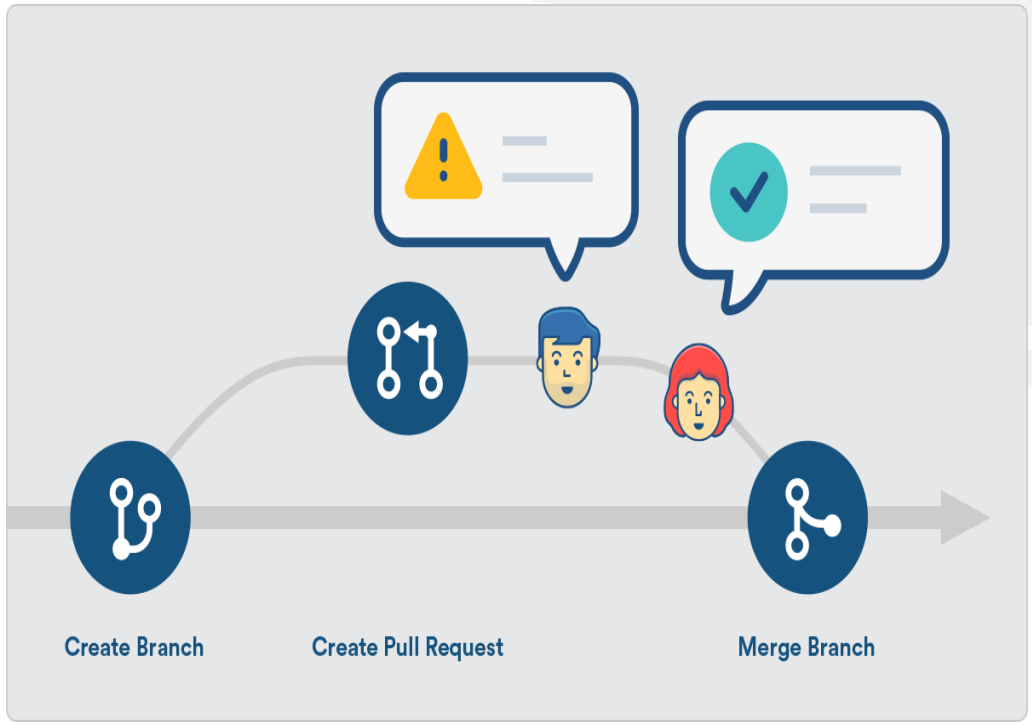
- **Authorization checks for each request to deny any URL manipulation kind of requests:** A security principle that ensures that authority is not circumvented in subsequent requests of an object by a subject, by checking for authorization (rights and privileges) upon every request for the object.
- In other words, the access requests by a subject for an object are completely mediated every time.
- All accesses to objects must be checked to ensure that they are allowed.

Few Important Add-ons

- **Maintaining versions**: Make it a habit to display the version number somewhere in the application. This is helpful in identifying version-specific issues.



Few Important Add-ons

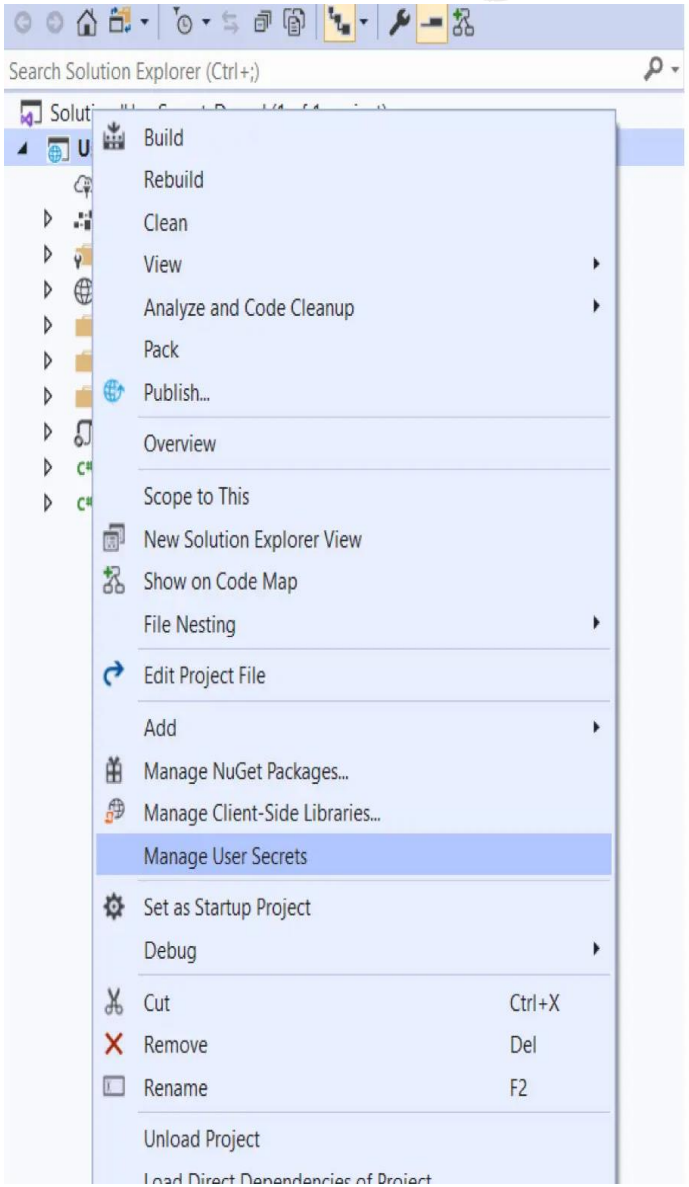


- **Back-end Calculations:** All calculations of payment totals, billed amounts, etc. are done within the API or back-end code and never accepted from the front-end.
- **Branch Management:** It allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.
- **Divide and Implement:** Divide the task into multiple smaller chunks rather than look at it as a whole and then start the implementation.

Few Important Add-ons

- **UserSecrets.json:** Avoid uploading environment variables with source code. use secret.json to save environment variables in local environment

```
{
  "ConnectionString": "This is a test connection string",
  "APIKey": "This is s secret key",
  "AppSettings": {
    "GlobalSettings": {
      "GlobalAccessKey": "This is a global access key!"
    }
  }
}
```



THANK YOU!