Coding Best Practices

While writing any piece of code, its very important to remember that you will not be the only person working on it forever. So it's your job to write it in such a way that the next person will be able to understand and build upon it with least amount of hassle.

Having said that, here are some guidelines that we must follow to ensure best coding practices:

1. Consistent Formatting and Indentation:

- Maintain a consistent coding style throughout your project. All files should be uniform in their formatting, should not have any extra spaces between the code lines. A tool like Prettier should be used to automatically format the code neatly.
- Use consistent indentation (usually 2 or 4 spaces). A lot of times we come across code that is not properly indented, this creates huge readability issues as the person working on the code finds it hard to understand where one function ends and the other begins.
- Choose a naming convention for variables, functions, and classes and stick to it (e.g., camelCase, snake_case, PascalCase). It is recommended that all variables should have name in snake_case, all functions in camelCase and all files PascalCase.

2. Meaningful Naming:

- Use descriptive names for variables, functions, classes, and methods. No more var1, tt_1, tt_2 type of naming. If a variable is going to contain name of a user, it should be user_name or name.
- Names should convey the purpose and functionality of the entity clearly and without any ambiguity.

3. Modularization:

- Divide your code into smaller, reusable modules or functions. Restrict the size of your files and functions to as small as possible. In today's world where we have enormous computational power, the size of the files make little impact on the compilation and execution speed of the code, however in a large application the impact adds up and makes the entire application significantly slower. **As a rule of thumb, the code should only have as many lines as necessary**.

4. Comments and Documentation:

- Add comments to explain complex logic, algorithms, or any non-obvious behaviour. Comments will help other developers understand what a particular piece of code is supposed to do and can also be used to caution them about potential breakages that might happen if the code is changed.

5. Error Handling:

- Every function **MUST** have error handling implemented. Always handle negative cases of a variable not being present or a function not returning the right response and handle the errors gracefully.
- Provide informative error messages in logs to aid in debugging. For server side scripting languages like PHP, logs are very important and there should be descriptive logs for any kind of

error or exception handling that we implement. A log saying "failed" or "error here" is not very useful, whereas if the log says "failed because user returned value 123" then we can debug using this message and make the code better in future iterations.

6. DRY (Don't Repeat Yourself) Principle:

- **NEVER DUPLICATE CODE.**
- If you ever come across a use case where you are writing the same code again, in the same application but in a different place, you need to figure out how to use a common function to do the job. It's a very bad practice to duplicate code in 10 different places because in the future when you make a change in 1 place, you need to change in all places and if by human error you don't change everywhere or if you are new to the codebase and don't know where all to make the change, you will end up introducing new bugs to the system.

7. Optimization:

- As a rule of thumb: **Optimize code for readability first; then for performance.**
- More money, time and resources is spent maintaining the code than running it. So it's absolutely necessary that we optimize for readability first and for performance second.

8. Security:

- Be mindful of security vulnerabilities like SQL injection, cross-site scripting (XSS), and others right when you are writing the code for the first time.
- Let's say you are creating a registration form, you should create the UI, apply validations in the UI, sanitize inputs in the controller, write a common insert function in the model using query builder and then use this function in the controller to insert while handling the case for a failed query and returning the relevant error message. All these steps are necessary and if you write it in this way, the code will be much more secure than simply creating UI, sending data to controller, creating a new insert function in model and then writing a raw query to insert. While both approaches work, the first one creates a high quality software with minimum vulnerabilities and high maintainability, the second approach creates the impression of a sloppy job.

9. Avoid Global State:

- Minimize the use of global variables. They can lead to unintended side effects and make code harder to understand and debug.
- Instead of defining global variables, you can just as easily use the env file to get data across the application.

10. Code Reviews:

- Its necessary to have periodic code reviews to ensure compliance to the best coding practices. As a way of practice, we should follow a maker-checker principle, where the maker writes the code and any other person in the team checks his code to ensure compliance.

11. Keep Dependencies Minimal:

- Only include dependencies that are necessary for your project. While installing new packages, first check if the package has any vulnerabilities and if it does, then try to find a work around or a new package that can do the job just as well.
- Regularly update dependencies to benefit from bug fixes and improvements that the

maintainers of the package make.
- Make sure you remove any and all dependencies that are not required for the application anymore.

12. Keep It Simple:
- Aim for simplicity and avoid unnecessary complexity. As engineers, the only way we know how to flex is by displaying our ability to comprehend the complex things easily. This is a futile exercise when writing code. As a coder, your job is to write the **MOST EASILY MAINTAINABLE CODE** that you can and **NOT** the most complex.

13. Plan and Design:
- Spend time designing your code before jumping into implementation. A pen and paper is your best friend for writing the best code. This might sound archaic and counter intuitive but it's true.
- Whenever you are working on a new functionality or starting implementation of a complex feature, its advisable to spend 5-10 mins planning the input, processing and output of the function. Once you plan it on paper, you will realise that your job of coding is reduced to just typing as you don't have to think about the logic anymore.

Remember that coding practices vary depending on the programming language and the specific project, so adapt these guidelines as needed. Consistency and communication within your team are key to successfully implementing and maintaining these best practices.

If someone points out an issue in your coding practice or suggests an improvement, always take the criticism positively.

Remember: **We cannot get better without positive feedback!**

Revision #1
Created 10 August 2023 05:28:07 by Nilesh Rathour
Updated 10 August 2023 06:05:27 by Nilesh Rathour