Q2)

1. Make Your Log Entries Meaningful With Context

Add as much context as possible to your log messages with respect to their level and expected granularity. That will not only help its readers to get a headstart on fixing the problem. But it will also make each message more unique and easily distinguishable.

2. Use a Standard Date and Time Format

The timestamp is an essential piece of information that should be included in every log entry. It's no use to know that something happened without knowing when it happened. But how should you include this information? That can be a tricky decision since there are many date/time formats out there.

The best practice is to use a format that is universally understandable and free from ambiguities. The good news is that such format exists and it's an ISO standard called ISO-8601. There are some different variations of the format, but its advantages are present in all of them. In short, by using this format you avoid confusion.

3. Use Local Time + Offset for Your Timestamps

Some people argue that you should only use UTC when logging, which isn't bad advice (it does solve the problems above.) But today we're mainly talking about human-readability when it comes to log files. Users will talk in terms of their timezone (e.g., it was about 2pm EST when the problem happened). Thus, it might make more sense for the timestamps to be in users' timezone. That makes it easier for the developer performing post-mortem debug to find the relevant log entries. But how do you solve the DST and timezones problems, while keeping the timestamps in the user's local time?

Well, that's one of those rare cases when you can have your cake and eat it too. Just use the local time, along with the offset from UTC. That way, it becomes easier for locals to understand when an issue happened relative to their own time. The offset allows you to convert the timestamp back to UTC easily—invert its signal and add it to the hour.

4. Employ Logging Levels Correctly

Properly employing logging levels is one of the most important things you can do for your log files. That'll make it easier for automatic parsers to understand them. But it also helps human readers, which is our main focus today. The correct use of levels will make your log files more easily searchable and, yes, readable. But what are logging levels?

Briefly, logging levels are labels. You use them to categorize your log entries by *urgency*, or *severity*. By applying logging levels, you gain the ability to easily search and filter your log entries by their levels. Levels also help you control the granularity of your log entries—more on that later.

So, what are the best practices regarding logging levels?

First, always add the log level to your log entries. Additionally, make sure to display the level in uppercase (ERROR instead of "error.") That will make the level easily distinguishable from the other information on the entry. Finally, use the appropriate level for each event**.** For instance, don't use ERROR for a typical, expected event—for example, if a user logged on the system.

5. Split Your Logging to Different Targets Based on Their Granularity

When it comes to production, on the other hand, it makes sense to log fewer events. For instance, you might log the actions of the user that are important from a business logic perspective, but not each interaction the user had with the UI. For instance, you would usually log “User X created a new project template” instead of “User X opened the new project template screen; User X clicked on the “Add new template” button…” and so on.

So, the advice here is log to different targets, based on level/granularity. “Target” here means the log’s destination. It might be a database table, a text file, and so on. When you’re searching through a log, trying to understand what went wrong in production, you don’t want to be bogged down by thousands of debugging entries.

6. Include the Stack Trace When Logging an Exception

This section should be short since its title pretty much says everything there is to it. When logging an exception, you should always include its stack trace. For the developer performing a post-mortem debug, the stack trace is essential information that will help them connect the dots.

7. Include the Name of the Thread When Logging From a Multi-Threaded Application

Like the previous section, this one also has its contents spoiled by its title. When logging from an application with multiple threads, always include the name of the thread where the event happened. That will enable searching/filtering, making the entries not only easily parseable but also more readable for humans.

8. Make the Logs Human-Readable as Well

Log files should be machine-parsable, no doubt about that. But they should also be human-readable as well. Why is that?

Simply put, people will read the log entries. And those will probably be (somewhat) stressed-out developers, trying to troubleshoot a faulty application. Don’t make their lives harder than they have to be by writing log entries that are hard to read.

OK, but how do we achieve human-readable logs? Well, the Scalyr blog has an entire post covering just that, but here are the main tidbits:

- Use a standard date and time format (ISO8601)
- Add timestamps either in UTC or local time plus offset
- Employ log levels correctly
- Split logs of different levels to different targets to control their granularity
- Include the stack trace when logging exceptions
- Include the thread’s name when logging from a multi-threaded application

9. Avoid Vendor Lock-In

This tip was already partially covered by the first one, but I think it’s worth mentioning it in a more explicit manner.

So, the advice here is simple: avoid being locked to any specific vendor. Organize your logging strategy in such a way that, should the need arise, it becomes simple to swap a logging library or framework with another one.

How do you do that?

One way is to make sure your application code doesn’t mention the third-party tool explicitly by making use of a wrapper. Rather, create a logger interface with the appropriate methods, and a class that implements it. Then, add to this class the code that actually calls the third-party tool. That way, you protect your application from the third-party tool. If you ever need to replace it with another one, just a single place has to change in the whole application.

In order to make this approach easier, you can adopt a logging façade, such as slf4j, which the post already mentioned. This project offers a standardized abstraction over several logging frameworks, making it very easy to swap one for another.

10. Don’t Log Sensitive Information

Finally, a logging security tip: don’t log sensitive information. First, the obvious bits. Make sure you never log:

- Passwords
- Credit card numbers
- Social security numbers

Now, the not so obvious things you shouldn’t log.

- Session identifiers Information the user has opted out of
- Authorization tokens
- PII (Personal Identifiable Information, such as personal names)

Also, you have to make sure you’re not inadvertently breaking the law. There are laws and regulations that prohibit you from recording certain pieces of information. The most famous of such regulation is probably GDPR but it isn’t the only one. Make sure you know and follow the laws and regulations from your country and region.

Why log formatting is necessary?

Log formatting is crucial to achieving readable log files. But why would you need readable (i.e., human-readable) log files in the first place?

You might argue that a log file should be machine-readable. I whole-heartedly agree with that. Having machine-readable logs enables you to use tools that can automatically parse, process, and analyze them. As a result, you can gain valuable insights that would otherwise be lost to you.

But your log files need to be easily readable to people, too. One of the primary purposes of logging is to enable post-mortem debugging. People will look at the log entries. They'll use the information they see there to try and reconstruct what happened with the application. That job is already hard—there's no need to make it harder by having unreadable log files.