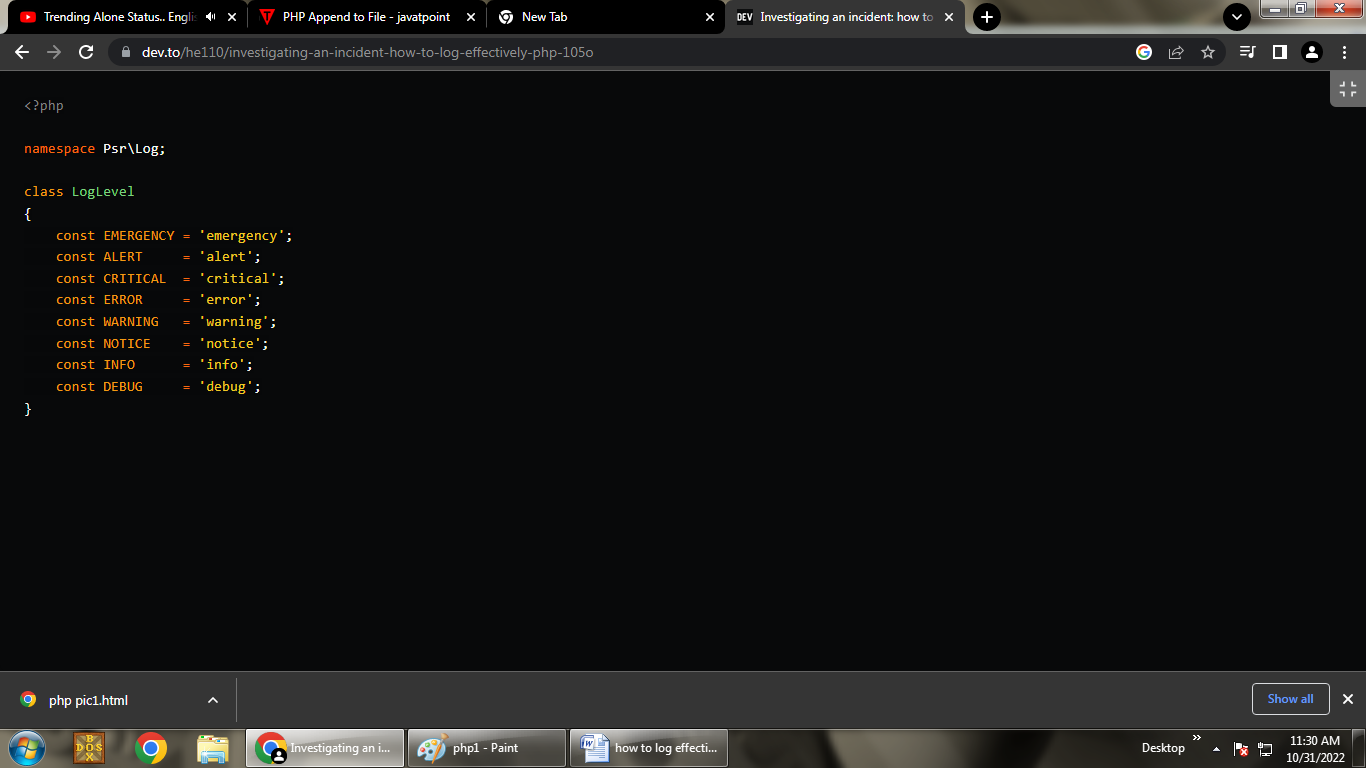
**ADVANCED WEB APPLICATION DEVELOPMENT**

**Module-5:**

**Task:** Implement an effective Logging System using file in PHP.

# how to log effectively (PHP)

PSR is a collection of general recommendations for PHP developers. It contains tips for code style, some interfaces, and other recommendations. One of its documents ([PSR-3](https://www.php-fig.org/psr/psr-3/)) is dedicated to a logger implementation.



PSR-3 defines 8 different message levels. Proper usage of them will simplify searching for errors and will boost up an accident reaction speed. Let's go deeper and figure out when to use them.

* **DEBUG** – debugging information that reveals the details of the event in detail;
* **INFO** – any interesting events. For instance: user has signed in;
* **NOTICE** – important events within the expected behavior;
* **WARNING** – exceptional cases which are still not errors. For example usage of a deprecated method or wrong API request;
* **ERROR** – errors to be monitored, but which don't require an urgent fixing;
* **CRITICAL** – critical state or an event. For instance: unavailability of a component or an unhandled exception;
* **ALERT** – error and an event to be solved in the shortest time. For example, the database is unavailable;
* **EMERGENCY** – whole app/system is completely out of order.

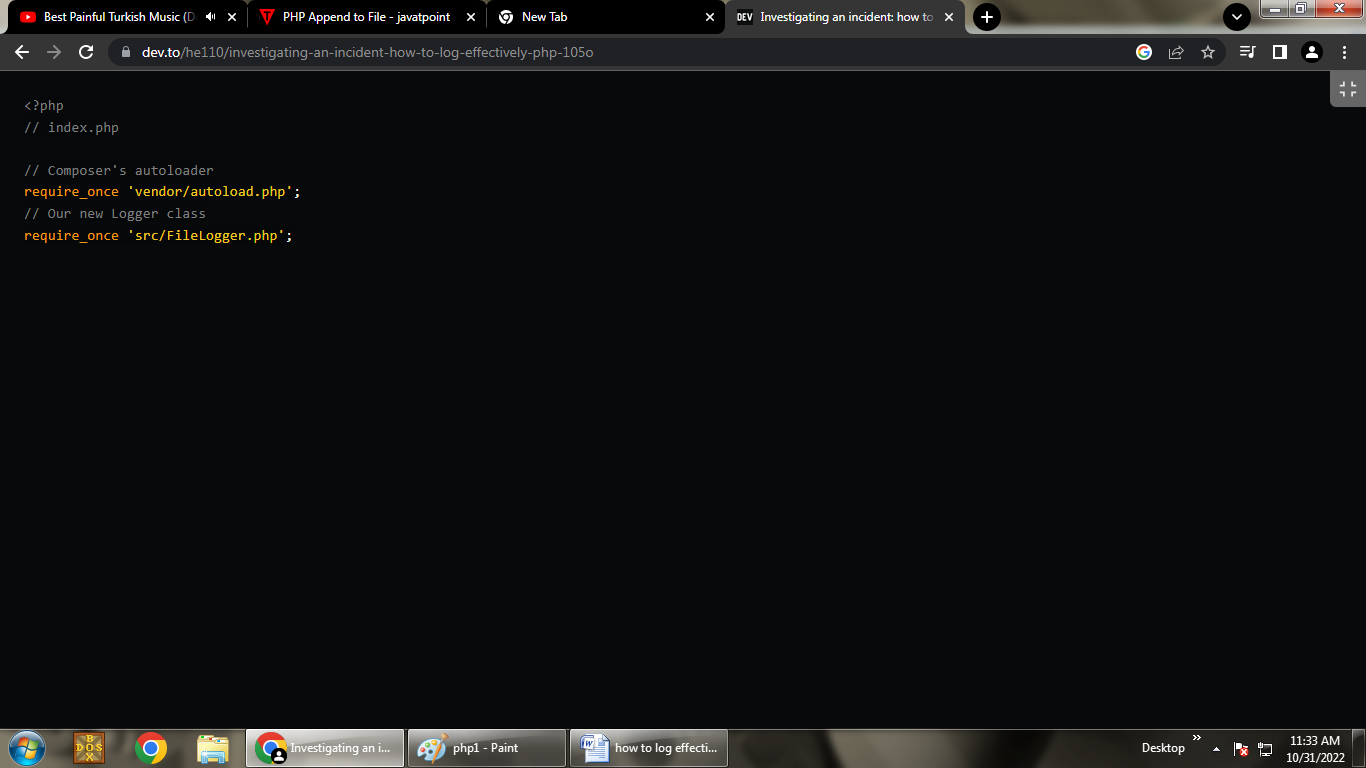
**PSR-3. Logger Interface**

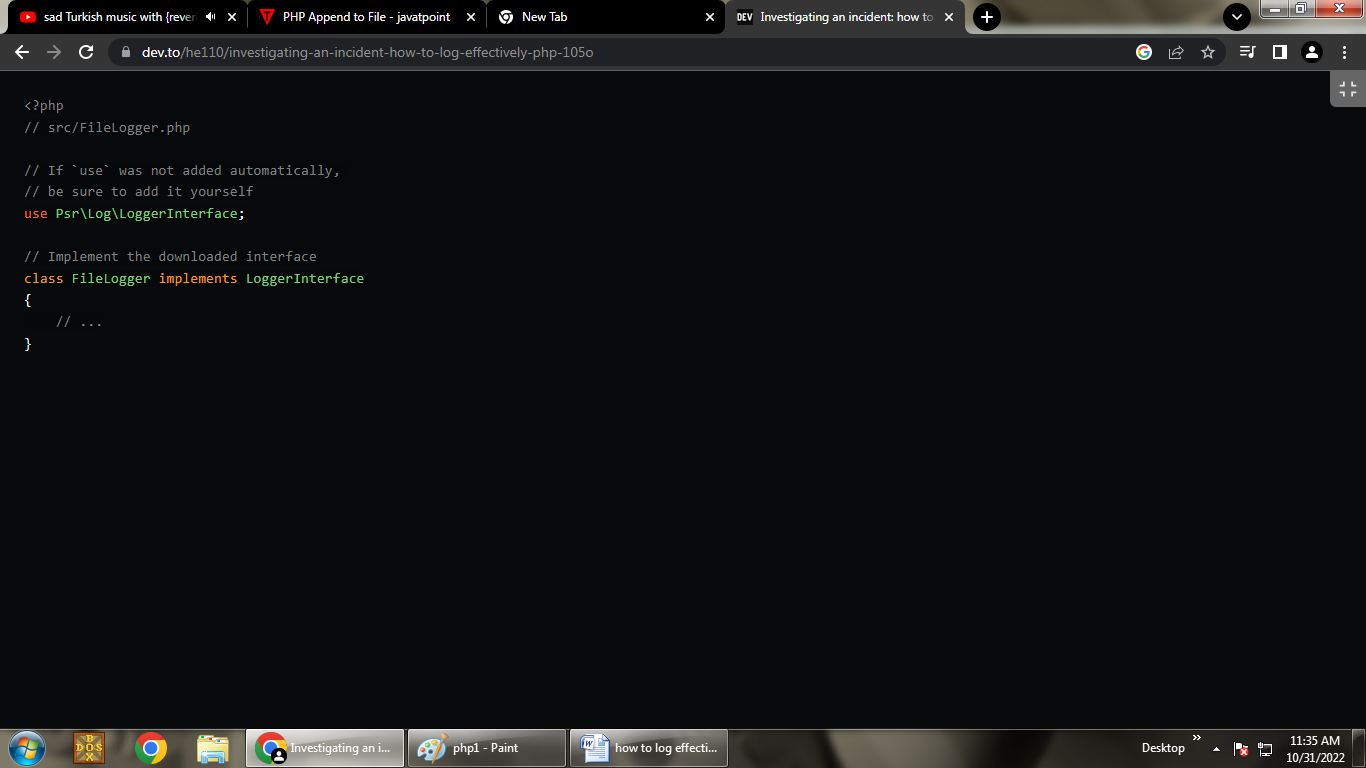
In addition to the class with levels, PSR-3 offers us an interface for implementing our own loggers - LoggerInterface. Its implementation is very useful because most existing libraries support it, and if you decide to replace your logger with another one, just connect a new class instead.

LoggerInterface requires the implementation of logging methods, in accordance with the previously reviewed levels.

composer req psr/log

The package we downloaded contains several classes, traits, and interfaces, among which the LogLevel we previously reviewed, and the LoggerInterface that is currently of our interest. Let's create a new class that implements that interface.





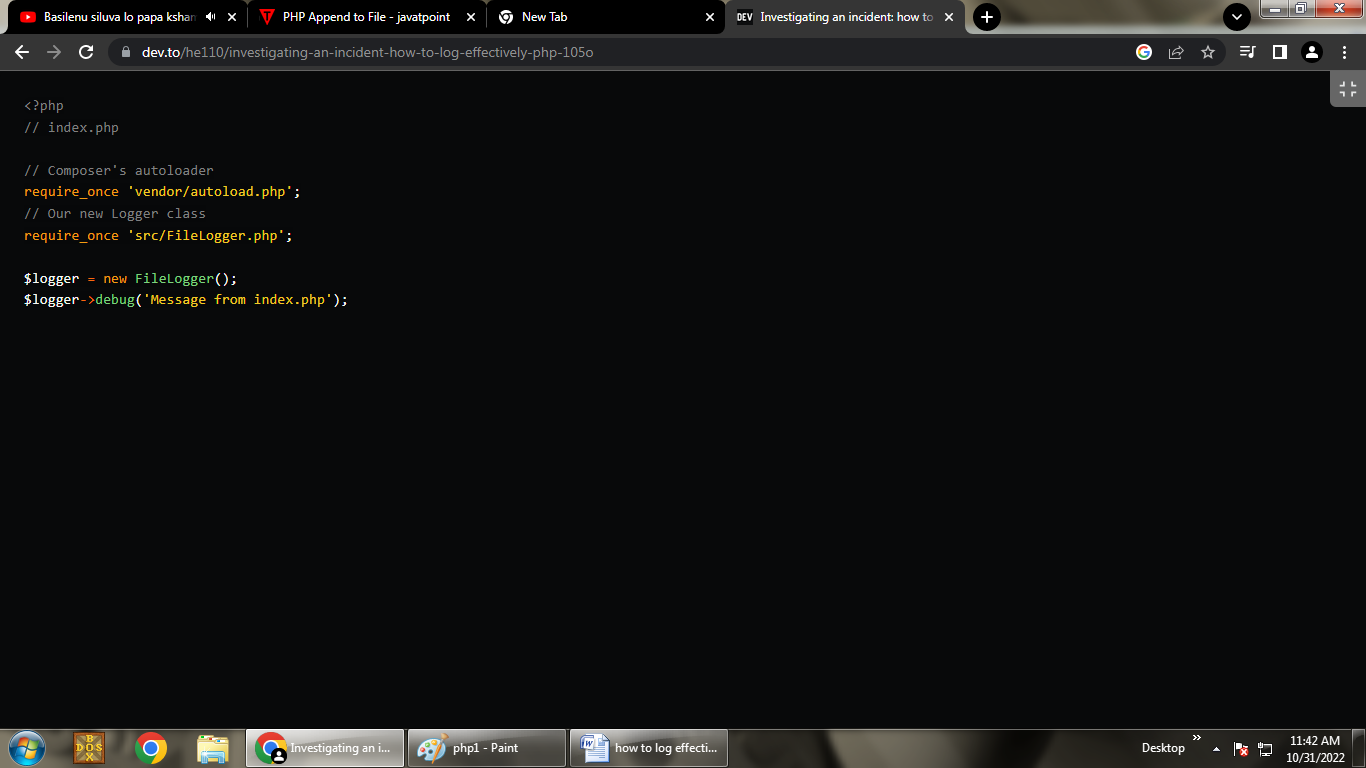
We created the class, but in order for it to meet the requirements of the standard, you need to write all the methods described in the interface.

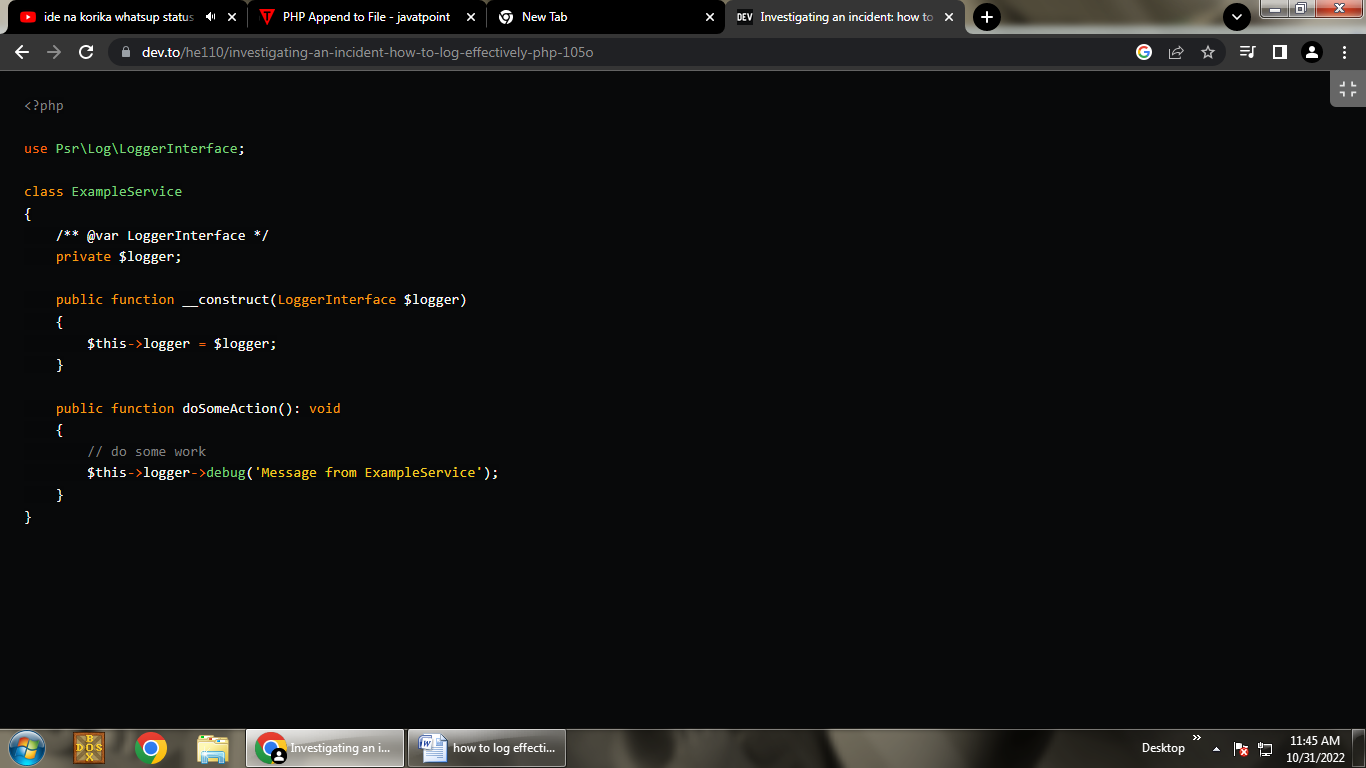




**Use of logger**

Now that our class implements the interface proposed by the PSR-3 standard, we can easily use it anywhere. For example, in the file **index.php**:

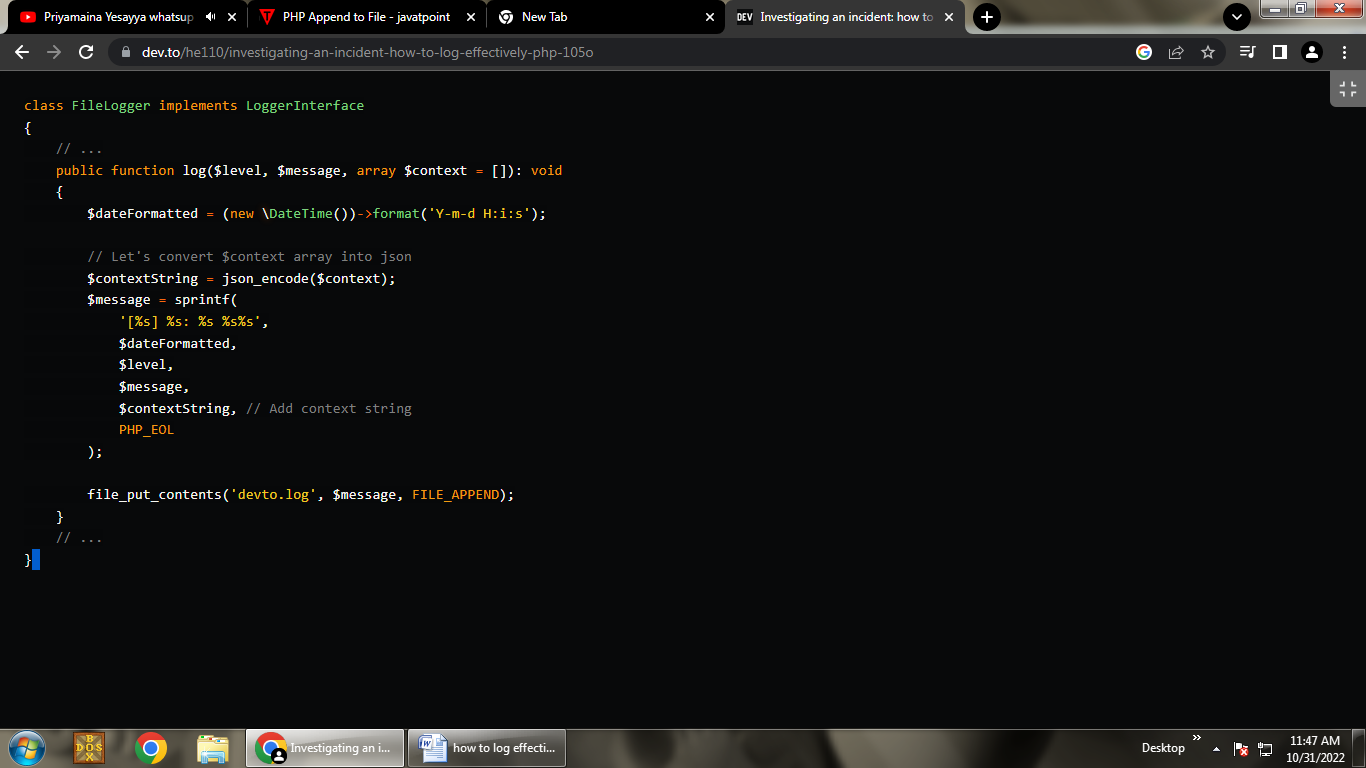




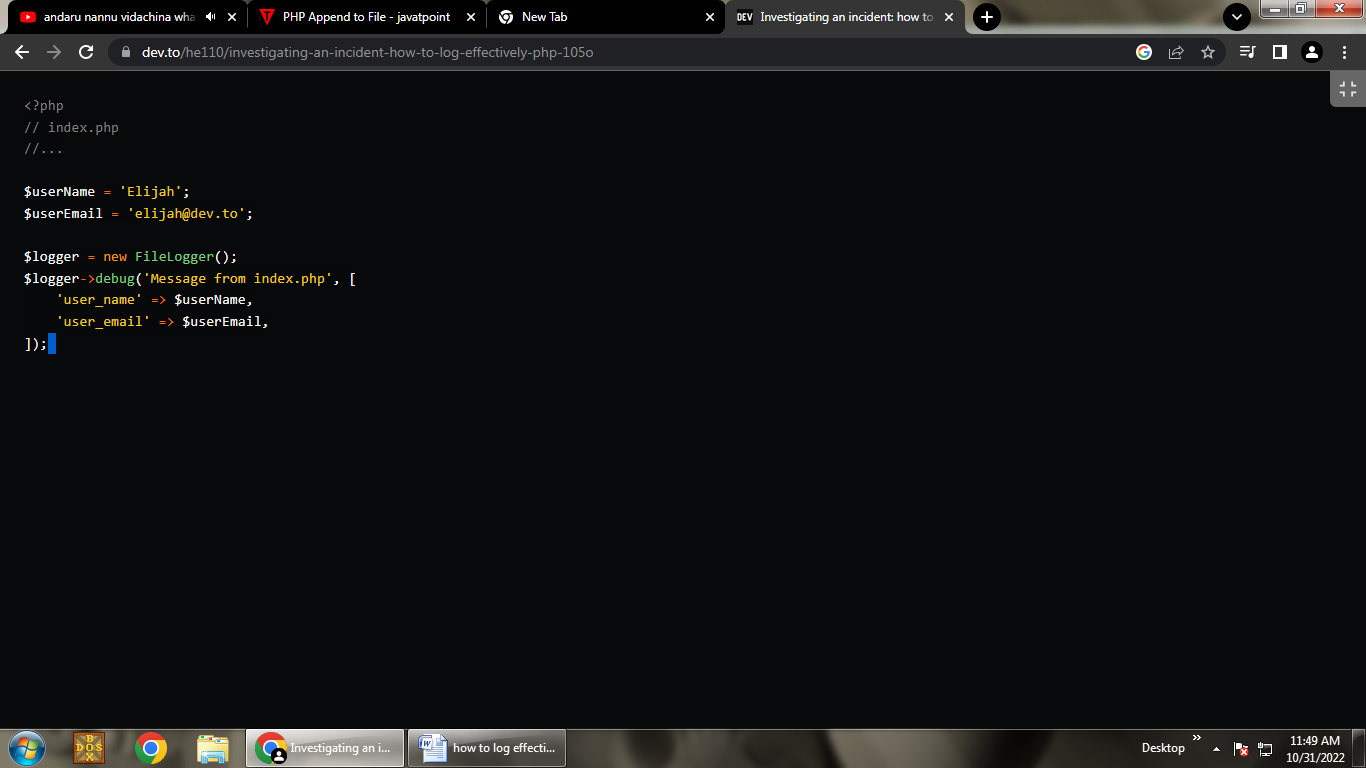
Please note that we do not specify the final implementation (FileLogger) as the type of the constructor argument, but the interface of the **PSR-3** standard. This is very convenient because it allows us to very easily replace the used logger with any other one that supports this interface.

**Context**

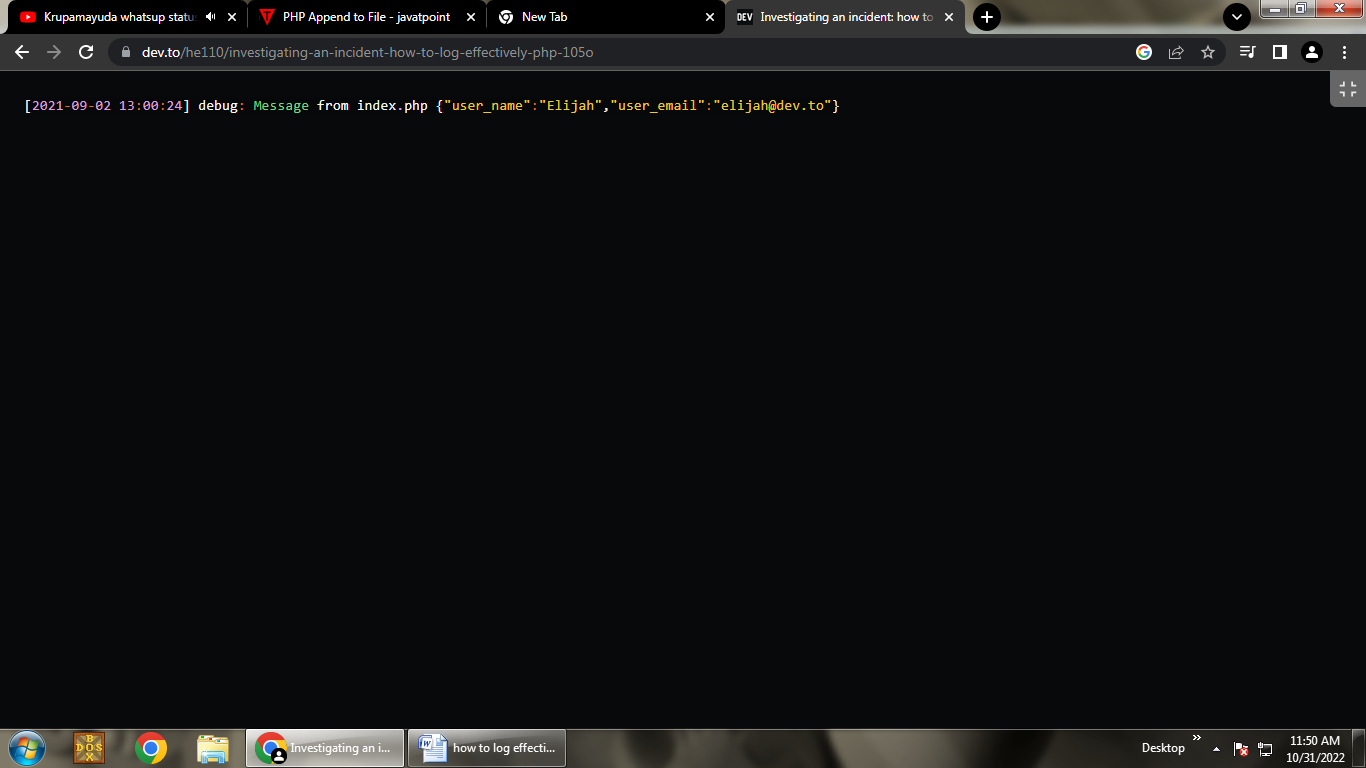
You may have noticed that all methods of the LoggerInterface contain the $context argument. Why is it needed?



Now, at any place of the logger call, we can pass an array of additional information with the second argument.



As a result, we will get a record of the following form:



Regarding the context, there is one very simple rule: any dynamic information should be transmitted in it, but not in the message.