**Laravel strategies, optimizations and best practices  
  
The purpose of this session -**

1. Not have others pull the rug under from your feet as a developer
2. Discuss what a consistent structure for any project should look like and how it can help

**Performance-**

1. Use select to limit the amount of data being fetched
2. Use joins instead of “with” to fetch data when both parent and child data is needed, and use “whereHas” instead of joins to filter parent data without the child data
3. Use arrays whenever possible instead of collections
4. Caching(i.e php artisan optimize)

**Security-**

1. Use try-catch with custom exceptions
2. Do NOT pass $request->all() to mass assignment
3. Use more validation rules
4. Do not use Integers in query params and ID fields

**Best Practices-**

1. Use versioning
2. SOLID principles
3. DRY Code(i.e abstracting common functionalities to separate functions)
4. Use of types(return type and function parameters)

**Popular design patterns in Laravel-**

1. Builder pattern (i.e query builder)

(example - https://programmingdive.com/how-to-use-method-chaining-in-php/)

1. Factory pattern (i.e returning views from controller)
2. Strategy pattern ()
3. Provider pattern (A lot of things in laravel i.e Eloquent, Translation, Cache, Config provider)
4. Repository pattern (Based on Interface pattern. Model should not be responsible for fetching data so we abstract data calls into a separate class where the calls are defined as functions.) (Just use scopes)
5. Facade pattern (Binding a class to a facade so it becomes available to all everywhere in the application)
6. Adapter pattern
7. Service pattern
8. Action pattern
9. Job/Event listener pattern
10. Observer pattern

**SOLID principles(How to use them in Laravel) -**

1. S - (1) Move your code away from controller to a separate class (2) Move validation logic to Form Request (if possible) or use a private function (3) Use query scopes in model
2. O - Use inheritance to extended classes
3. L - Use interfaces
4. I - If an implementation/class needs to have some functions and not all functions of an interface/class, separate out the functions to another interface/class instead of using some and ignoring/throwing exceptions for others
5. D - Pass interfaces to functions instead of passing concrete classes(i.e dependency injections), and make the passed classes extend the interface