Fasada

Jeśli znacie mnie, nie tylko z tego kursu, ale również z YouTube, instagrama czy innych sieci społecznościowych, dobrze wiecie że obecnie najbliżej mi do Laravela.

Laravel fasadami stoi i za to często jest krytykowany.

I zanim zaczniemy się zastanawiać, jak ten wzorzec można wykorzystać, chcę zacząć od przykładowej Laravela. Gdy Ottwell tworzył ten framework jego cel był dość jasny - programowanie ma być szybkie, łatwe i przyjemne. W świecie CRUDa, faktycznie, taki jest Laravel.

Problem polega jednak na tym, że wiele klas, takich choćby jak fasady, to obiekty boskie Laravela.

I to właśnie największy problem, jaki może Ci się przydarzyć, wdrażając ten wzorzec. Podobnie jak singleton, wzorzec ten może być dobry, ale i bardzo zły.

Mówiliśmy tutaj o tym, że do wszystkiego trzeba podchodzić z głową i tak samo jest właśnie z fasadą. Nie możemy pozwolić, by stała się boskim obiektem. Wówczas cała nasza aplikacja będzie z tą fasadą sprzężona.

Nie mniej jednak, gdy chcesz odizolować kod od złożoności systemu, fasada może okazać się dobrzym wyborem.

Fasada stanowi interfejs dla złożonego podsystemu, który może składać się z wielu małych części. Weźmy taki Storage z laravela. Możemy w nim ustawić dysk lokalny, ftpa, aws'ową s3 czy napisać swoje własne rodzaje dysków. Możemy też w ramach testów skorzystać z mockowania. Dzięki temu zamiast zapisywać pliki na dyskach, będziemy mogli udawać, że to się wydarzyło. Skomplikowana logika zapisu, odczytu, aktualizacji i innych elementów dzieje się już za Fasadą.

Przydatne. Trzeba tylko pomyśleć, czy nie jest to pewnego rodzaju usprawiedliwienie kodu, nie zgodnego z SOLIDem. Czy za pliki nie powinna być odpowiedzialna klasa wstrzyknięta do systemu? Fasada oczywiście zadziała podobnie, ale musi być tą konkretną klasą. Jeśli wspólnie umówimy się, że jest

to akceptowalne - co robimy, wybierając famework - to ok. Ważne jednak zdawać sobie sprawę z konsekwencji tych decyzji.

Ok, dość prawienia morałów. Zmierzmy się z naszym zadaniem.

Jest ono bardzo proste. Chcę włączyć światło.

I co, tylko tyle?

No w praktyce brzmi to łatwo, ale chodzi mi o światło do video. Po pierwsze mamy tutaj 3 lampy. Główną, taką dużą z softboxem, dodatkową, kontrową, świecącą na moją prawą stronę oraz dodatkową, kolorową lampę służącą za “uciekawianie tła”. Problem polega jednak na tym, że jeśli jest jasno - to te światła nie działają i nie da się nimi sterować w odpowiedni sposób.

W studiu musi być ciemno. Gdy jest późno, nie jest to problem, ale gdy chcemy nagrywać coś o 12:00 przydało by się zasłonić okno. Tylko wiecie, ja jestem programistą - nie mam zamiaru wstawać, skoro automatyczne zasłony mogą mi zrobić “noc”, światła mogą włączyć się same, a ja mogę po prostu nagrać dla was nową lekcję.

Na szczęście producent automatycznego kinkietu udostępnia swoje API. Lampy również możemy włączyć, dzięki temu że podpięte są do kontaktu IoT. Jedna z nich do takiego kontaktu podpięta nie jest, tutaj mamy specjalne API.

Dużo? No dużo - a my tylko chcemy włączyć światło.

I teraz fasada może być bardzo eleganckim obiektem, prostym w użyciu i nie pytających o zbyt wiele. W takiej sytuacji wymagać będzie jednak odpowiedniej konfiguracji. My tą sytuację na początek uprościmy.

Od producenta naszych zabawek dostaliśmy następujące interfejsy:

```
interface Light
{
    public function turnOn(): void;
    public function turnOff(): void;
    public function setPower(int $power): void;
}

interface Curtain
{
    public function open(): void;
```

```
    public function close(): void;
}
```

Oraz oczywiście ich implementację

```
class LightPower implements Light
{
    public function __construct(private string $powerName)
    {
    }

    public function turnOn(): void
    {
        echo "Turn on Light at power ".$this->powerName."\n";
    }

    public function turnOff(): void
    {
        echo "Turn off Light at power ".$this->powerName."\n"
    }

    public function setPower(int $power): void
    {
        echo "Update Light power at ".$this->powerName." for
    }
}

class WixLed implements Light
{
    public function turnOn(): void
    {
        echo "Turn on Wix LED\n";
    }

    public function turnOff(): void
    {
        echo "Turn off Wix LED\n";
    }
```

```
}

class CurtainRemote implements Curtain
{
    public function open(): void
    {
        echo "Open courtine\n";
    }

    public function close(): void
    {
        echo "Close courtine\n";
    }
}
```

Poza tym, stworzyliśmy jeszcze rozwiązanie, które pobiera informacje o tym czy słońce zaszło, czy jeszcze nie.

```
interface Weather
{
    public function isSunny(): bool;
}

class WeatherStation implements Weather
{
    public function isSunny(): bool
    {
        return rand(0, 1);
    }
}
```

Sporo tego, ale właśnie fasada, pozwoli nam to połączyć w jedną całość, tak by możliwe było korzystanie z systemu w łatwy sposób. Zaprogramujmy więc naszą klasę:

```
class Studio
{
    /**
```

```
     * @var Light[] $lights
     */
    public function __construct(
        protected array   $lights,
        protected Curtain $curtains,
        protected Weather $weather
    ) {
    }

    public function on(): void
    {
        if($this->weather->isSunny()) {
            $this->curtains->close();
        }

        foreach($this->lights as $light) {
            $light->turnOn();
        }
    }

    public function off(): void
    {
        $this->curtains->open();
        foreach($this->lights as $light) {
            $light->turnOff();
        }
    }

    public function setLightPower(int $power): void
    {
        foreach($this->lights as $light) {
            $light->setPower($power);
        }
    }
}
```

To po prostu obsługa wzajemnie powiązanych ze sobą elementów. Jeśli jest słonecznie, uruchamiamy zasłony. Następnie włączamy wszystkie lampy i

gotowe.

Przy wyłączeniu, nie sprawdzamy już pogody - wyłączamy wszystko i odsłaniamy okna.

Fasada pozwala nam również na ustawienie mocy światła. Oczywiście również w tym wypadku można by się pokusić o implementację innych zależności - np. jeśli jest jasno a my mamy odsłonięte okno to moc będzie powiększona itd. itd.

Aby użyć tego kodu należy uruchomić taki kod:

```
$lights = [
    new LightPower('Listwa 220V'),
    new LightPower('Kontrowe 220V'),
    new WixLed()
];
$curtain = new CurtainRemote();
$weather = new WeatherStation();

$studio = new Studio($lights, $curtain, $weather);
$studio->on();
$studio->setLightPower(10);
```

Ale, jak mówiłem wcześniej, Laravel przyzwyczaił nas do nieco “wygodniejszego” działania z fasadami. Obecne rozwiązanie nie zapewnia nam również możliwości “udawania” że obiekt jest wykorzystywany.

Teraz wyobraź sobie, że przy każdym teście jednostkowym, Twoja roleta w pokoju zaczyna hałasować, a światła robią jedno wielkie disco. Nie chcemy tego. Dlatego teraz zastanowimy się jak ten interfejs można by uprościć.

Powiedzmy, że chciałbym mieć możliwość uruchomienia studia za pomocą takiego kodu:

```
StudioFacade::on();
```

A następnie zmiany światła za pomocą

```
StudioFacade::setLightPower(10);
```

I tu pojawia się problem. By było to możliwe, musielibyśmy użyć wzorca Singleton, który już poznałeś. Tak naprawdę, cała fasada stała by się boskim

obiektem, którego można używać na przekroju całego systemu.

Właśnie za to Laravel najczęściej jest krytykowany.

I mimo że nie jest to dobre rozwiązanie, pokażę wam, jak można by było je wdrożyć, w następnej lekcji.